Universidad Nacional Autónoma de México

Facultad de Estudios Superiores Aragón

Ampliación y mejoramiento de funcionalidades a un sistema controlador de iluminación RGB destinado para el apoyo de estímulo multisensorial de niños con discapacidades cognitivas.

T E S I S

Que para obtener el título de:

INGENIERO ELÉCTRICO ELECTRÓNICO

Presenta:

Mauricio Guzmán Peña

Director de tesis:

Ing. Juan Manuel Hernández Contreras

Nezahualcóyotl, Estado de México, 2022

Agradecimientos

A mi familia, por su apoyo constante e incondicional a través de mi carrera académica y en mi vida.

A los ingenieros Juan Manuel Hernández Contreras y Alejandro Andrés Serapio Carmona, así como al Doctor Ismael Díaz Rangel, por la guía ofrecida a lo largo del desarrollo de este trabajo.

A la Universidad Nacional Autónoma de México, por permitirme formar parte de su comunidad y brindarme las herramientas y conocimientos para ser alguien productivo para la sociedad.

Contenido

Índice de figuras

Índice de tablas

Capítulo 1 Introducción

El ser humano es un espécimen animal que pertenece a la especie “Homo Sapiens”, el cual se caracteriza por su capacidad de raciocinio y desarrollo de diferentes niveles de intelecto, lo cual le permite adquirir una amplia gama de conocimientos. Pese a ello, existen alteraciones que perjudican estas cualidades y que pueden darse por cierta variedad de factores: Accidentes durante el desarrollo del individuo, o problemas en el periodo de gestación o anomalías en las células reproductoras al momento de la fecundación, ya sea en el óvulo o el espermatozoide. Estos factores llevan a los humanos a padecer trastornos congénitos.

La parálisis cerebral infantil (PCI) es un grupo de trastornos que afectan la capacidad de una persona para moverse y mantener el equilibrio y la postura. La parálisis cerebral es causada por el desarrollo anormal del cerebro, o por un daño al cerebro en etapa de desarrollo, que afecta la capacidad del niño para controlar sus músculos. Existen varios motivos posibles que dan lugar a estas causas (Anónimo, 2020).

Otro trastorno congénito es el Síndrome de Down. Este se presenta cuando un individuo tiene 47 cromosomas, en vez de los 46 usuales. Tener un cromosoma extra cambia la forma en que el cuerpo y el cerebro se desarrolla, de ahí que el Síndrome de Down tenga efectos sobre las características físicas, la salud y el aprendizaje.

Los trastornos congénitos son, en muchos países, causas importantes de mortalidad infantil, enfermedad crónica y discapacidad. Los trastornos congénitos graves más frecuentes son las malformaciones cardiacas, los defectos del tubo neural y el Síndrome de Down.

Estos trastornos pueden afectar a los bebés independientemente de dónde nazcan, de su etnia o de su raza. Cada año, entre el 3 y 6% de bebés en el mundo nacen con un defecto de nacimiento grave. Aquellos que viven con estas afecciones están en mayor riesgo de tener discapacidades de por vida. En México, se estima que 11,339 niños y 8,428 niñas nacieron con algún defecto de nacimiento en el 2016.

Los datos de la Organización Panamericana y Mundial de la Salud (OPS y OMS) revelan que la mayoría de los casos de trastornos congénitos son prevenibles o tratables (Secretaría de Salud, 2018).

México cuenta con el Centro de Rehabilitación e Inclusión Infantil Teletón (CRIIT), que es un sistema privado de rehabilitación y sin fines de lucro. Aquí se atiende a las necesidades de niños y adolescentes con diferentes discapacidades, entre ellas el síndrome de Down, la parálisis cerebral, el autismo y el cáncer.

Dentro las instalaciones del CRIIT Nezahualcóyotl se cuenta con espacios para la atención médica y psicológica, rehabilitación pulmonar, terapia de lenguaje, fisioterapia, terapia ocupacional, hidroterapia, cuartos para la estimulación temprana de los sentidos y el desarrollo de las facultades cognitivas para los niños que nacieron o adquirieron a causa de un accidente alguna de estas condiciones.

Para dar apoyo a las terapias, se establecieron pláticas entre personas de la carrera de Ingeniería Eléctrica-Electrónica de la FES Aragón y terapeutas del CRIIT, derivando en la propuesta de crear un sistema electrónico que permitiese que los niños con discapacidades cognitivas puedan identificar la causa-efecto de acciones, ello basándose en lo observado en el cuarto de estimulación multisensorial. El proyecto se ha desarrollado en etapas; el primer prototipo creado es un sistema que cuenta con cinco dispositivos denominados “módulos”. De estos cinco módulos, cuatro se denominan transmisores y uno es el receptor (figura 1.1). Dichos transmisores consisten en unos dispositivos que tiene un cilindro con seis “manubrios” de colores diferentes, y que pueden ser manipulados por lo niños. Su funcionamiento consiste en que envíen su orientación con respecto del suelo, considerando tres ejes y seis posiciones, al módulo receptor; cada posición tiene un color asociado (rojo, verde, azul, cian, magenta y amarillo), y el receptor hará que unas tiras LED emitan el color correspondiente. En este sistema, los cuatro emisores pueden trabajar de manera simultánea e independiente.

El intercambio de información se hace a través de radiofrecuencia, lo que permite tener un alcance de varias decenas de metros entre los transmisores y el receptor.

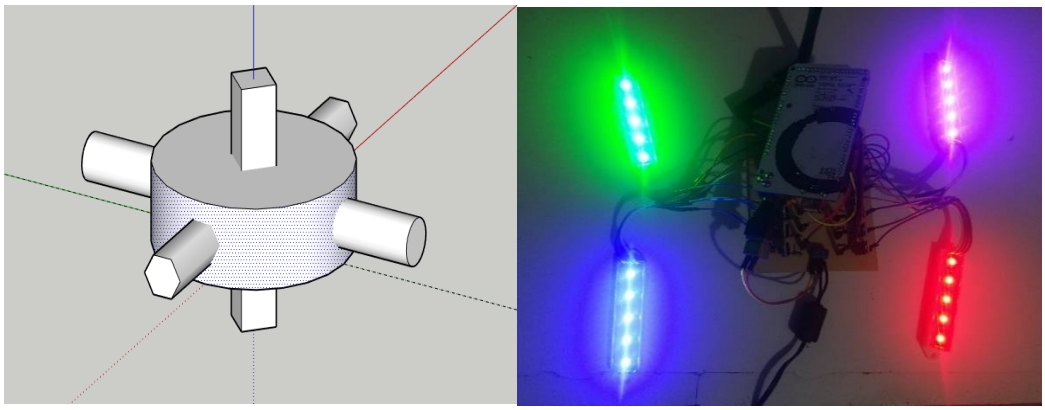

Figura 1.1 Prototipo del transmisor (izquierda) y del recetor (derecha).

En este trabajo se da continuidad al desarrollo del sistema incorporando algunas mejoras, como lo es la adición de un zumbador a los transmisores para que se efectúe un sonido diferente en cada cambio de posición, así como la incorporación de baterías recargables y capacidad de carga inalámbrica de las mismas, con el propósito de que en un trabajo futuro se puedan utilizar estos dispositivos en el área de hidroterapia. También, se incorpora una interfaz para teléfonos celulares para monitorear el nivel de carga de las baterías, así como la opción de habilitar-deshabilitar los zumbadores. Otra mejora incorporada, consiste en el aumento de capacidad en la etapa de potencia, para permitir el uso de tiras LED RGB de hasta tres metros de largo para cada módulo.

1.1 Objetivo general

Aumentar y mejorar las características de un sistema de control de iluminación RGB por posicionamiento en tres ejes, que tiene el propósito de ser incorporado como parte de los elementos de estimulación multisensorial en el Centro de Rehabilitación e Inclusión Infantil Teletón, del municipio de Nezahualcóyotl.

1.2 Objetivos particulares

- Incorporar la capacidad de emitir sonidos distintivos en los módulos transmisores en función de su orientación.
- Reevaluar las opciones de acelerómetros/giroscopios y, de ser el caso, adaptar la opción más conveniente.
- Cambiar la etapa de suministro de energía de los transmisores por una con opción a baterías recargables y carga inalámbrica.

- Modificar la etapa de potencia para dar la capacidad de alimentar cuatro tiras LED RGB de hasta tres metros cada una.
- Diseñar una aplicación para teléfonos celulares Android, y modificar el módulo receptor para conectarse a la aplicación, la cual debe permitir activar-desactivar los zumbadores de los módulos transmisores, y conocer la cantidad de carga de sus baterías.

1.3 Actividades

- Revisar dispositivos capaces de emitir sonido.
- Analizar características funcionales y operativas de acelerómetros y/o giroscopios presentes en el mercado, para seleccionar una mejor opción en caso de ser necesario.
- Estudiar opciones de fuentes de alimentación basadas en baterías recargables y sistemas de carga inalámbrica.
- Identificar los requerimientos de potencia y seleccionar dispositivos para su gestión.
- Examinar dispositivos que permitan la comunicación vía Bluetooth con un microcontrolador.
- Diseñar una app móvil para activar y desactivar las señales sonoras de los módulos, así como para presentar el nivel de carga de los mismos.
- Realizar PCB para cada módulo.

1.4 Justificación

Siempre será importante tratar de ayudar a las personas que más lo necesitan, con el afán de tener un mundo más inclusivo, lo cual es el propósito de este trabajo. Además, buscar la vinculación de instituciones académicas entre el sector público y privado, permite que los académicos y estudiantes tengan un mayor y mejor acercamiento con las necesidades reales de la sociedad, de ahí la relevancia de este trabajo, el cual, debido a su complejidad, se debe llevar a cabo por etapas. En su primera etapa sirvió como acercamiento al sistema que se planea incorporar al CRIIT, pero aún es lejano a su versión definitiva, por lo que es necesario seguir desarrollándolo.

1.5 Antecedentes

(Casillas, L. Morán, & Meza-Kubo, 2018) realizaron una evaluación a un prototipo de bajo nivel que pretendía ser usado en términos de la identificación de escenarios presentados y la evocación de los recuerdos a través de los estímulos proporcionados a los participantes de su estudio, en el que se hace mención a la eficiencia de la terapia de estimulación multisensorial para generar resultados positivos en pacientes con discapacidad cognitiva.

(Robles Vallejos, 2020) concluyó que una aplicación para la estimulación multisensorial en un entorno con estímulos controlados dará al paciente la libertad de experimentar varias experiencias sensoriales y motoras, permitiendo la corrección de la coordinación oculomotora, del lenguaje, la percepción cognitiva y la función motora, fortaleciendo su desarrollo integral y la comunicación sensorial, refuerza el aprendizaje y su vínculo con el medio circundante, su rendimiento y su calidad de vida.

(Alvarado Alvarado & Prado Monge, 2019) incorporaron equipo electrónico como soporte a la estimulación multisensorial en niños de 2 a 5 años de edad, en donde se muestra que la integración de la estimulación multisensorial en conjunto con las TIC (tecnologías de la información y la comunicación) permite desarrollar sistemas educativos interactivos, que brindan soporte a la educación en niños que presentan algún tipo de necesidad educativa, siendo que esta esté o no ligada a una discapacidad.

(Gonzalez Gonzalez & Guachun Arias, 2018) desarrollaron un sistema embebido multisensorial enfocado a niños diagnosticados con parálisis cerebral y sus trastornos asociados. Si bien realizaron modificaciones para adaptar el sistema a las necesidades de los pacientes, su sistema puede ser utilizado por niños con diferentes tipos de discapacidades y en diferentes ambientes.

(Mateos Peña, 2018) desarrolla un sistema controlador de iluminación RGB por posicionamiento en tres ejes, con el propósito de ser usado por niños con algún trastorno congénito, en el área de hidroterapia, logrando obtener un prototipo funcional. Si bien su objetivo era más dejar las bases para un sistema cien por ciento

funcional que el comprobar que su prototipo brindaba mejoras a los pacientes con dicho padecimiento, sirve como punto de partida y permite enfocarse en implementar mejoras que permitan acercarse cada vez más al ideal que él planteó.

1.6 Organización del trabajo

El capítulo 1 muestra el panorama general del trabajo de investigación y el desarrollo del trabajo de tesis.

En el capítulo 2 se analiza la información recopilada para la comprensión del tema, tanto en el ámbito médico como el de la ingeniería.

El capítulo 3 aborda el desarrollo del sistema y su método de operación desde la programación hasta el diseño del controlador de iluminación.

En el capítulo 4 se presentan las pruebas realizadas al sistema en sus diferentes etapas y los resultados obtenidos del análisis de cada una.

Por último, se muestran las conclusiones del proyecto, el trabajo a futuro y la integración de la documentación anexa utilizada en el proyecto.

Capítulo 2 Marco teórico

2.1 Trastorno congénito

También denominado como defecto de nacimiento, anomalía congénita o malformaciones congénitas (figura 2.1), son anomalías estructurales o funcionales, como los trastornos metabólicos, que ocurren durante la vida intrauterina y se detectan durante el embarazo, en el parto, o en algún momento posterior de la vida.

No ha sido posible asignar una causa específica a los trastornos congénitos. No obstante, se ha logrado identificar algunas causas o factores de riesgo, como lo son factores socioeconómicos y demográficos, factores genéticos, infecciones, estado nutricional de la madre, y factores ambientales (Organización Mundial de la Salud, 2020).

Figura 2.1 Trastorno congénito en un pie.

Un trastorno congénito puede afectar la apariencia del cuerpo, su funcionamiento, o ambos. Los trastornos congénitos pueden variar de leves a severos, y la forma en que éstos afectan la vida de un niño depende de qué órgano o parte del cuerpo está involucrado y qué tan serio es el problema.

A menudo, los niños con trastornos congénitos necesitan cuidados y tratamientos especiales. Dado que los síntomas y problemas causados por dichos trastornos

varían, los tratamientos también son diferentes. Las opciones de terapia pueden variar entre cirugía, medicamentos, dispositivos de asistencia, fisioterapia, y terapia del habla. Los niños con defectos de nacimiento pueden necesitar una variedad de servicios y es posible que requieran la atención de varios especialistas (Biblioteca Nacional de Medicina de los EE. UU., 2020).

2.2 Parálisis cerebral

La parálisis cerebral (figura 2.2) es la discapacidad motora más frecuente en la niñez. Es causada por el desarrollo anormal del cerebro o por un daño al cerebro en desarrollo, y afecta la capacidad del niño para controlar los músculos.

Figura 2.2 Niño con parálisis cerebral.

El daño al cerebro que lleva a padecer de PCI puede ocurrir antes del nacimiento, durante el parto, dentro del primer mes de vida, o durante los primeros años de vida, que es cuando el cerebro aún se encuentra en desarrollo. Cuando el daño al cerebro ocurre antes o durante el parto, se le llama parálisis cerebral infantil congénita, siendo que del 85 al 90% de los casos son congénitos. En un pequeño porcentaje la causa del daño al cerebro ocurriría 28 días después del parto. En este caso, se le llama parálisis cerebral infantil adquirida, y por lo general se asocia a una infección (como meningitis) o a una lesión en la cabeza.

Los síntomas varían de un caso a otro. Puede que una persona con esta afección necesite usar un equipo especial para poder caminar, o que caminar no le sea posible y necesite cuidados por el resto de su vida. En casos no tan graves, podrían caminar con dificultad, pero sin que sea realmente necesaria alguna clase de ayuda

especial. La parálisis cerebral no empeora con el tiempo, pero los síntomas que se presentaron en un principio pueden ir cambiando a lo largo de la vida de la persona.

No existe cura para la parálisis cerebral, pero el tratamiento puede hacer más a mena la vida de quienes padecen de esta afección. Es importante comenzar con un programa de tratamiento lo más pronto posible. Luego de hacer el diagnóstico, un grupo de profesionales de la salud trabajará con el niño y su familia con el propósito de establecer un plan para ayudar al niño a alcanzar su máximo potencial. Los tratamientos más comunes incluyen medicamentos, cirugía, aparatos ortopédicos, y terapia física, ocupacional y del habla (Anónimo, 2020).

2.2.1 Síndrome de Down

El Síndrome de Down (figura 2.3) se presenta en personas que nacen con una copia extra del cromosoma 21, de ahí que también se le conozca con el nombre de Trisomía 21. Las personas con este padecimiento pueden tener tanto problemas físicos como mentales, y en cada persona la afección es diferente (Biblioteca Nacional de Medicina de los EE. UU., 2020).

Figura 2.3 Niña con síndrome de Down.

El Síndrome de Down muestra rasgos físicos específicos en aquellos que lo padecen, como lo es el tono muscular bajo, baja estatura, ojos inclinados hacia arriba, y un solo pliegue profundo que pasa por el centro de la palma de la mano. Estas características físicas se dan en grados diferentes, o no se dan en absoluto (Sociedad Nacional del Síndrome de Down, 2020).

Las posibilidades de dar a luz a un bebé con Síndrome de Down aumentan mientras mayor sea la edad de la madre. El Síndrome de Down no tiene cura, por lo que una

atención temprana puede mostrar mejoras hasta cierto punto. Esta atención puede incluir terapia de lenguaje, física, ocupacional y/o educacional (Biblioteca Nacional de Medicina de los EE. UU., 2020).

Las personas con Síndrome de Down experimentan retrasos cognitivos, pero el retraso es generalmente de leve a moderado y no resulta ser signo de calificar a quien lo padece de “menos acto” para realizar alguna tarea. Los niños con Síndrome de Down aprenden a sentarse, caminar, hablar, jugar, y hacer gran variedad de actividades, sólo que un poco más tarde que otros niños sin Síndrome de Down (Sociedad Nacional del Síndrome de Down, 2020).

2.3 Estimulación sensorial y fisioterapia

La estimulación sensorial, también conocida como *snoezelen*, tiene origen en Holanda. Fue desarrollada en la década de los 70 por dos terapeutas holandesas. Tiene como fin potenciar el desarrollo de aquellos que acuden a ella, para ayudarles a enfrentar situaciones que les permitan mejorar su capacidad para interactuar con el medio y los elementos que se encuentran en el mismo, así como facilitar sus procesos de aprendizaje sociales a través de los cinco sentidos. Con el propósito de alcanzar dichos objetivos durante su infancia y en siguientes etapas de su vida se han creado espacios multisensoriales (figura 2.4) donde los niños son expuestos a estímulos controlados que les permiten percibir distintas sensaciones que les ayudarán a adquirir y conocer el aprendizaje por medio del descubrimiento (Balsells, 2017).

La estimulación sensorial o multisensorial plantea que el mundo en el que vivimos está lleno de sensaciones producidas por la luz, el sonido, el olor, el gusto y el tacto, a los que podemos percibir a través de nuestros órganos sensoriales (nariz, boca, ojos, oídos, piel). Son diversas las investigaciones publicadas que tienen como objetivo comprobar el cómo las personas con alguna discapacidad intelectual pueden mejorar sus capacidades en base a una estimulación de sus capacidades sensoriales (Cid Rodríguez & Camps Llauradó, 2010).

Figura 2.4 Panel de estimulación sensorial.

La fisioterapia, o terapia física, es una profesión sanitaria y universitaria, centrada en el movimiento y la función humanos, y en maximizar su potencial. Utiliza técnicas físicas para favorecer, conservar y restaurar el bienestar físico, psicológico y social, considerando las variantes en el estado de salud, como se puede ver en la figura 2.5.

Entre las técnicas más conocidas se encuentra la terapia manual. Resulta incorrecto pensar que la terapia manual consta sólo de masajes, pues realmente se trata de cualquier acción que involucre el uso de las manos para tratar algún problema de salud. Así, la terapia manual puede resultar en masajes, movilizaciones y manipulaciones articulares, neuronales, enfocada al aparato respiratorio, linfáticas, faciales, puntos gatillo, etc. Existen otro tipo de técnicas, como el ejercicio terapéutico, educación terapéutica, reducción postural, y punción seca (agujas), entre otras más. De entre estas, en el ejercicio terapéutico se usan técnicas en piscinas, técnicas de estabilidad, equilibrio, máquinas o mecanoterapia, integración de yoga, pilates o cualquier otro tipo de movimiento que lleve a la mejora del paciente. La reducción postural, que también cuenta con sinergias y patrones neuro-motores patológicos, se usan en pacientes con problemas neurológicos, como en niños con parálisis cerebral, lesionados medulares, esclerosis múltiple, etc (Sanz, 2019).

Figura 2.5 Sesión de fisioterapia.

Los fisioterapeutas combinan técnicas para mejorar la coordinación, la fuerza, la resistencia, la flexibilidad y la amplitud del movimiento. Es común que los fisioterapeutas recurran a bicicletas o caminadoras para tratar a sus pacientes. También puede tratar a sus pacientes con frío o calor, con estimulación eléctrica, ultrasonido, masajes, o terapia acuática (ejercicios en piscina). Al igual que los médicos, los fisioterapeutas se especializan en diferentes áreas. Por ejemplo, un fisioterapeuta podría enfocarse a tratar lesiones deportivas; algún otro preferirá especializarse en lesiones en la cabeza, o en daños musculares y cuidado de lesiones dadas por quemaduras o daños en la piel (Nemours, 2017).

2.3.1 El Cuarto de Estimulación Multisensorial (CEMS)

El cuarto de estimulación multisensorial (CEMS) es un área física destinada a la estimulación del sistema nervioso central a través del uso de diversos materiales y herramientas, el acompañamiento psicológico a las familias, y el asesoramiento por parte de pedagogos. Siendo exactos, cuando el cuarto está adecuado para estimular más de un sentido se le conoce como cuarto multisensorial, el cual, se puede dividir en varios espacios, cada uno enfocado a la estimulación de un sentido (figura 2.6). La recepción y la asimilación de estímulos externos como lo serían los olores, sabores, colores y superficies apoyan tanto a la ubicación espacio-tiempo, como al aprendizaje (Molina Velázquez & Banguero Millán, 2008).

Figura 2.6 Cuarto de estimulación multisensorial (CEMS).

El cuarto cuenta con diferentes objetos y dispositivos para estimular los sentidos. Algunos de estos objetos pueden ser tableros de opciones, para que el paciente obtenga un aprendizaje visual de la experiencia táctil; paneles con sonidos agradables, que también se puede llevar en compañía de algunas otras modalidades a la relajación del paciente y a mejorar la comodidad e interacción; aromas que estimulen al ser percibidos y diferenciados como olores representativos; diferentes figuras, estructuras y texturas que, a través de la manipulación, logran mejorar la habilidad motora del paciente, así como su control voluntario, fuerza muscular, movimientos finos, y su coordinación viso-motora (Fisiolab, 2020).

2.3.2 La hidroterapia

La hidroterapia, o terapia acuática, es un método terapéutico utilizado por la fisioterapia y brinda incontables beneficios a distintas patologías del aparato locomotor. En la hidroterapia se emplean conocimientos fisioterapéuticos de rehabilitación y biomecánica junto con diversas técnicas de tratamiento, que buscan conseguir el mayor aprovechamiento de las propiedades y bondades que ofrece el medio acuático. La hidroterapia es un complemento del proceso de rehabilitación, por lo que también puede emplearse más técnicas fuera del agua que permitan alcanzar una mejor eficiencia en el tratamiento.

Las sesiones de hidroterapia (figura 2.7) son llevadas a cabo de acuerdo a las necesidades que deban ser cubiertas para cada caso particular de los pacientes

posterior a su evaluación, ya que no todo proceso de rehabilitación es una técnica generalizada, sino que debe ser adaptado a cada persona, a la patología y a la sintomatología (Sanchez, 2020).

Figura 2.7 Sesión de hidroterapia.

Un ciclo de hidroterapia suele realizarse de dos a tres sesiones por semana, y su duración dependerá de la condición del paciente. Cabe mencionar que los efectos curativos de la hidroterapia no son atribuibles al agua en sí. Por el contrario, se deben al efecto en el cuerpo por parte de los estímulos térmicos (frío y calor), químicos (uso de preparados en el agua), y mecánicos (mayor o menor presión en las extremidades). Además del hecho de que aun cuando en un curso de rehabilitación el paciente realiza ejercicios similares a los que haría en un gimnasio, un programa de hidroterapia correcto debe regirse por tres principios básicos: la viscosidad del agua, la presión hidrostática, y la flotación.

Para poder llevar a cabo las sesiones de hidroterapia, antes debe ser consultado con un experto en fisioterapia que, después de una evaluación del estado físico del paciente, y de los objetivos que se ha fijado previamente, le recomendará un programa de rehabilitación determinado (TopDoctors, 2020).

2.4 LED RGB e iluminación policromática

2.4.1 Luz

La luz es una onda electromagnética que puede ser percibida por el ojo humano, y cuya frecuencia determina su color. La luminotecnia es la ciencia que estudia las principales formas de producir luz, su control y sus aplicaciones.

La luz es una fracción del espectro electromagnético que puede ser visto por el ojo humano. Se le llama espectro electromagnético al conjunto de ondas electromagnéticas que emite o absorbe un objeto, y esta radiación sirve para determinar al objeto. Se puede clasificar al espectro electromagnético de acuerdo con su longitud de onda (figura 2.8). En este caso, y partiendo desde los de menor longitud de onda, tenemos a los rayos cósmicos, los rayos Gamma y los rayos X, seguidos por la luz ultravioleta, la luz visible y los rayos infrarrojos, hasta las ondas de radio, que son las de mayor longitud de onda.

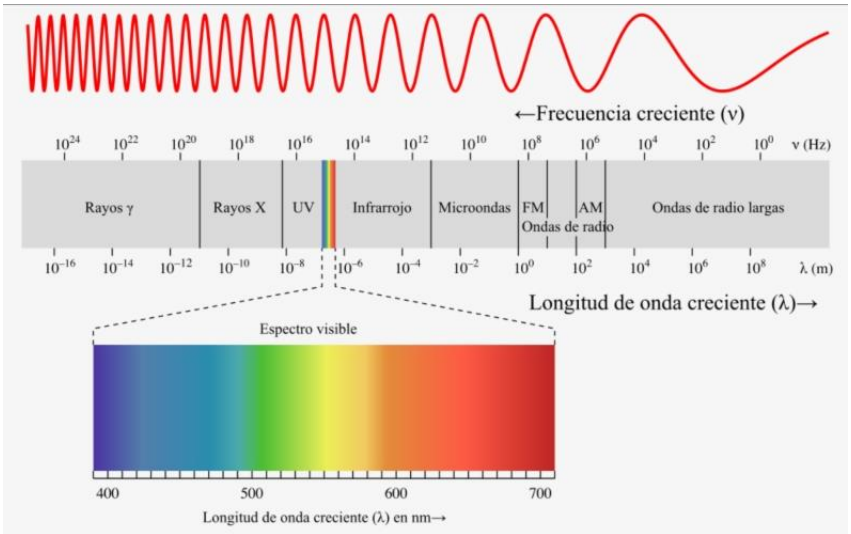

Figura 2.8 Espectro de luz visible.

La luz que es visible forma parte de una pequeña franja, que va desde longitudes de onda del violeta (380 nm) al rojo (780 nm). Los colores del espectro se ordenan de la misma forma que en el arco iris, lo que da lugar al espectro de luz visible. Es en esta franja del espectro donde las ondas electromagnéticas interactúan (son reflejadas o absorbidas) con la materia, y nos permite ver los objetos, su forma y su posición.

También podemos definir qué frecuencia o conjunto de frecuencias es emitida o reflejada por el objeto; es decir, el color que tiene. La luz blanca es producida cuando todas las ondas del espectro de luz visible se encuentran presentes en proporciones e intensidades iguales.

Una fuente o espectro de luz se considera monocromático cuando éste consta de radiación de una sola longitud de onda; esto es tomado como el espectro más discreto posible. Por lo tanto, una mezcla de luces monocromáticas roja y verde resultará en una luz monocromática amarilla. Ya que una luz monocromática no puede ser una mezcla, lo que se hace es generar algo que al ojo humano le parece idéntico; es aquí donde el concepto de color aparece (Optica INAOE, 2020).

2.4.2 Modelo de color RGB

Es un modelo de color aditivo en el que se agregan luz roja, verde y azul de diferentes formas para producir una variedad de colores, como se ve en la figura 2.9. El nombre del modelo viene de las iniciales de los tres colores primarios aditivos: rojo, verde y azul (en inglés: Red, Green, Blue).

El objetivo principal de este modelo es la detección, representación y visualización de imágenes en sistemas electrónicos, como lo son principalmente televisores y computadoras, pero también ha sido utilizado en la fotografía convencional y otras aplicaciones. El modelo es dependiente del dispositivo en el que se está empleando. Diferentes dispositivos reproducen un valor RGB determinado de forma diferente.

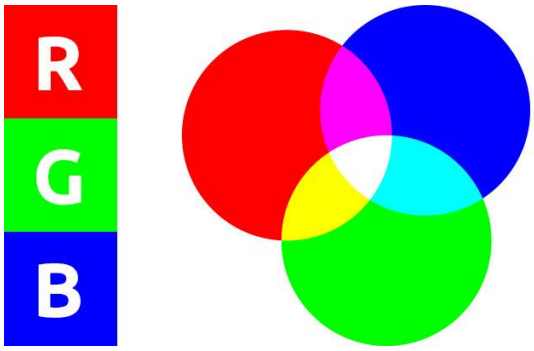

Figura 2.9 Modelo de colores RGB.

Para formar un color con el modelo RGB se deben superponer tres haces de luz. Cada uno de estos haces es denominado componente de ese color, y cada uno de estos componentes puede tener una intensidad arbitraria en la mezcla, desde totalmente apagado, hasta totalmente encendido. La intensidad cero en los tres componentes da el color negro, y la intensidad máxima en los tres da blanco; la calidad de este blanco dependerá de la naturaleza de las fuentes principales de luz. Cuando las intensidades son distintas dan como resultado un tono coloreado más

o menos saturado, dependiendo de la diferencia de las intensidades más fuertes y más débiles de los colores primarios empleados.

Cuando uno de los componentes tiene una intensidad superior a los otros dos el color resultante es un matiz próximo a ese color primario (rojizo, verdoso o azulado), y cuando son dos los componentes que tienen una intensidad superior el resultante es un matiz de un color secundario (cian, magenta o amarillo).

Un color secundario es formado por la suma de dos colores primarios de igual intensidad; verde con azul resulta en cian, rojo con azul resulta en magenta, y verde con rojo resulta en amarillo. Cada color secundario es el complemento de un color primario; esto es que, cuando un color primario y su color secundario complementario se suman, el resultado es el color blanco (Hisour, 2020).

2.4.3 Diodo Emisor de Luz (LED)

Los LED son diodos semiconductores que son capaces de emitir luz al pasar corriente eléctrica a través de ellos. El nombre LED proviene de las siglas correspondientes al acrónimo en inglés de *Light Emitting Diode*, traducido al español como *Diodo Emisor de Luz*.

Los LED se conforman por un chip de material semiconductor dopado con impurezas, y estas impurezas crean conjunciones del tipo P-N (figura 2.10). En comparación con los dispositivos luminarios más conocidos (focos), los LED poseen polaridad, por lo que sólo funcionarán si se encuentran polarizados de forma correcta. La polaridad está definida por el ánodo (terminal positiva) y el cátodo (terminal negativa).

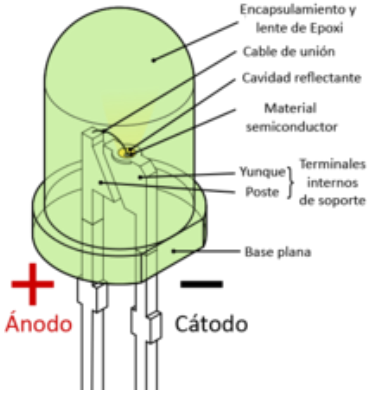

Figura 2.10 Esquema de un LED.

Al estimular las terminales del diodo con un voltaje, las cargas eléctricas negativas (electrones) y las cargas eléctricas positivas (protones) son atraídas a la zona de conjunción, donde se combinan entre sí, resultando en la liberación de energía en forma de fotones. A este efecto se le conoce con el nombre de electroluminiscencia (Leds International, 2020).

2.4.4 LED RGB

El led RGB es un tipo especial de LED compuesto por tres matrices LED simples, como las que se encuentran dentro de los LED monocromáticos. Con estas tres matrices pueden generar los tres colores primarios, y dar lugar a una variedad de colores diferentes con tan sólo controlar una de las terminales de este componente.

Las terminales de estos LED son un poco diferentes a los LED monocromáticos, pues tienen tres pines (terminales) que corresponden al cátodo (positivo) de cada uno, y un pin más que corresponde al ánodo, y es común a todos (negativo), como se puede ver en la figura 2.11. Debido al tipo de material semiconductor del que están hechos es que se puede conseguir esa variedad de colores (Isaac, 2020).

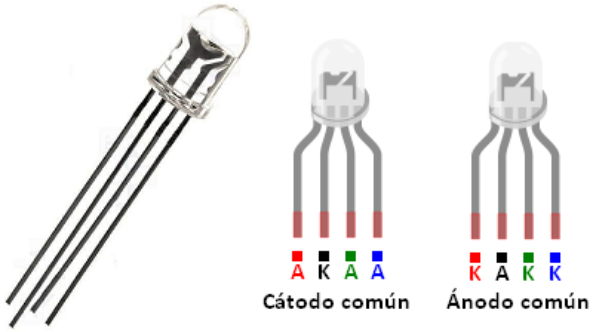

Figura 2.11 LED RGB.

2.4.5 Relación de colores en código decimal

El modelo de color RGB no define por sí mismo exactamente qué es el rojo, verde o azul, por lo que los mismos valores RGB pueden mostrar colores considerablemente diferentes entre diferentes dispositivos que utilicen este modelo de color.

Para indicar la intensidad con la que se hará uso de un color en la mezcla se asigna un valor a cada uno de los colores (figura 2.12), de manera que el valor ‘0’ significará que no formará parte de la mezcla, y a medida que el valor aumente, se debe

entender que formará en mayor medida parte de la mezcla. Si bien el intervalo de valores podría ser cualquiera, en informática se codifica a cada color primario con un byte (8 bits). Así, la intensidad de cada componente se mide en una escala que va del 0 al 255.

Color	Valor RGB
Negro	rgb(0, 0, 0)
Blanco	rgb(255, 255, 255)
Rojo	rgb(255, 0, 0)
Azul	rgb(0, 0, 255)
Verde	rgb(0, 255, 0)
Amarillo	rgb(255, 255, 0)
Magenta	rgb(255, 0, 255)
Cian	rgb(0, 255, 255)
Violeta	rgb(136, 0, 255)
Naranja	rgb(255, 136, 0)

Figura 2.12 Relación de colores en código decimal.

El rojo se obtiene usando la combinación (255,0,0), el azul se obtiene con (0,0,255), y el verde se obtiene con (0,255,0). Para obtener los colores secundarios, se hace uso de la combinación que tenga dos valores máximos, siendo (255,255,0) para el amarillo, (255,0,255) para el magenta, y (0,255,255) para el cian. Al optar por utilizar valores máximos y mínimos en los tres parámetros se obtiene el color blanco y el negro, respectivamente (Google Sites, 2020).

2.4.6 Tira de LED RGB modelo SMD 5050

El LED SMD (Surface Mount Device – Dispositivo de Montaje Superficial) es un chip muy pequeño envuelto en resina *epoxi* que, en forma de unidad o de tira (figura 2.13), se puede fijar en una superficie. Este tipo de LED es de material semiconductor (como nitruro de galio e índigo, o fosfuro de galio).

Se debe aplicar un voltaje de 2 Vcc a 3.6 Vcc a cada LED para que funcionen de forma correcta, con lo que se logra una corriente en el rango de 20 mA a 30 mA. En esas condiciones, la luz emitida por cada chip será, en promedio, de 5.5 lúmenes.

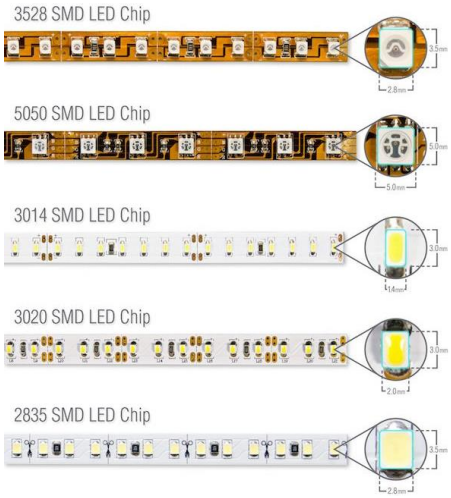

Figura 2.13 Diferentes formatos de tiras LED RGB.

Estas tiras de LED, independientemente de la marca, se comercializan en 30, 60 y 90 LED por metro, en forma estándar. Las hay también de 120 LED por metro, pero es prácticamente una tira doble que genera mucho calor y se comercializa sin protección o recubrimiento para que pueda disipar la carga térmica que genera.

Las tiras se encuentran de 3, 5 y 6 metros lineales, y se puede hacer un corte cada tres LED (cada 5 cm) o cada seis LED (cada 10 cm). El voltaje que debe proporcionar la fuente de alimentación es independiente del largo en que se corte la tira LED.

Los LED son sensibles al agua y a la abrasión producida por el polvo. Distintas tiras LED cuentan con diferentes medios de protección contra estos factores. Una forma internacional de indicar qué tan bien protegido está la tira LED es indicando el grado IP (Ingress Protection), que está formado por dos números. El primero indica el nivel de protección contra agua, y el segundo indica el nivel de protección contra el polvo. (Kohen, 2021)

2.5 Arduino

Arduino nace en el 2005 debido a la necesidad de contar con un dispositivo para utilizar en aulas en el Instituto de Diseño Interactivo de Ivrea (Italia), y que fuera de bajo costo. En la figura 2.14 se puede apreciar a los principales participantes de la idea y diseño de dicho dispositivo.

Arduino es una plataforma de desarrollo basada en una placa electrónica de hardware libre que incorpora un microcontrolador reprogramable y una serie de pines hembra. Estos pines permiten establecer conexiones entre el microcontrolador y diferentes sensores y actuadores de una manera muy sencilla. Una placa electrónica es una PCB (“Printed Circuit Board”; “Placa de Circuito Impreso” en español). Una PCB es la forma más compacta y estable de construir un circuito electrónico, por lo que la placa Arduino no es más que una PCB que implementa un determinado diseño de circuitería interna. De esta forma el usuario final no debe preocuparse por las conexiones eléctricas que necesita el microcontrolador para funcionar, y puede empezar directamente a desarrollar las diferentes aplicaciones electrónicas que necesite.

Figura 2.14 (De izquierda a derecha) David Cuartielles, Gianluca Martino, Tom Igoe, David Mellis, Massimo Banzi, equipo inicial de Arduino.

Cuando se habla de Arduino se debe especificar un modelo en concreto. Se han fabricado diferentes modelos de placas Arduino oficiales, cada una pensada con un propósito diferente y características variadas (tamaño físico, número de pines E/S, modelo del microcontrolador, etcétera). A pesar de las varias placas que existen, todas pertenecen a la misma familia (microcontroladores AVR marca Atmel). Esto significa que comparten la mayoría de sus características de software, como arquitectura, bibliotecas y documentación.

Arduino es libre y extensible, haciendo que cualquiera que desee ampliar y mejorar tanto el diseño hardware de las placas como el entorno de desarrollo, puede hacerlo

sin problemas. Esto permite que exista un rico ecosistema de placas electrónicas no oficiales para distintos propósitos, así como bibliotecas de software de terceros que pueden adaptarse mejor a nuestras necesidades.

Su entorno de programación es multiplataforma, por lo que puede instalar y ejecutar en sistemas operativos Windows, Mac OS y Linux. Su lenguaje de programación está basado en C++, que es un lenguaje de fácil compresión.

La placa Arduino estándar (Arduino UNO) tiene un valor aproximado de $250.00 (pesos mexicanos). Pero también está la opción de que uno mismo la puede construir (una de las ventajas del hardware libre), con lo que el precio de la placa podría incluso ser menor (Arduino, 2020).

2.5.1 Arduino Mega

El Arduino Mega 2560 es una placa de microcontrolador basada en el ATmega2560 (figura 2.15). Tiene 54 pines de entrada/salida digital (de los cuales 15 se pueden usar como salidas PWM), 16 entradas analógicas, 4 UART (puertos serie de hardware), un oscilador de cristal de 16 MHz, una conexión USB, un conector de alimentación, un encabezado ICSP, y un botón de reinicio. La tabla 2.1 muestra más de sus características.

Tabla 2.1 Datos sobre el Arduino Mega.

Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm

Cada uno de los 54 pines digitales del Arduino Mega se pueden utilizar como entrada o salida mediante las funciones pinMode (), digitalWrite () y digitalRead (), y operan a 5 voltios. Cada pin puede proporcionar o recibir 20 mA como condición de operación recomendada, y tiene una resistencia de pull-up interna (desconectada por defecto) de 20-50 k Ohms. Un máximo de 40 mA es el valor que no debe superarse para evitar daños permanentes al microcontrolador. Además, algunos pines tienen funciones especializadas:

- Serial: Pines 0 (RX) y 1 (TX); Serie 1: Pines19 (RX) y 18 (TX); Serial 2: Pines 17 (RX) y 16 (TX); Serial 3: Pines 15 (RX) y 14 (TX). Se utilizan para recibir (RX) y transmitir (TX) datos en serie TTL. Los pines 0 y 1 también están conectados a los pines correspondientes del chip serie ATmega16U2 USB a TTL.
- Interrupciones externas: Pines 2 (interrupción 0), 3 (interrupción 1), 18 (interrupción 5), 19 (interrupción 4), 20 (interrupción 3) y 21 (interrupción 2). Estos pines se pueden configurar para activar una interrupción en un nivel bajo, un flanco ascendente o descendente o un cambio de nivel.
- PWM: Pines 2 a 13 y 44 a 46. Proporcione una salida PWM de 8 bits con la función analogWrite ().
- SPI: Pines 50 (MISO), 51 (MOSI), 52 (SCK), 53 (SS). Estos pines admiten la comunicación SPI mediante la biblioteca SPI. Los pines SPI también están divididos en el encabezado ICSP, que es físicamente compatible con Arduino/Genuino Uno y las antiguas placas Duemilanove y Diecimila Arduino.
- LED: Pin 13. Hay un LED incorporado conectado al pin digital 13. Cuando el pin es de valor ALTO, el LED está encendido, cuando el pin es BAJO, está apagado.
- TWI: Pines 20 (SDA) y 21 (SCL). Admite la comunicación TWI utilizando la biblioteca Wire. Tenga en cuenta que estos pines no están en la misma ubicación que los pines TWI en las antiguas placas Duemilanove o Diecimila Arduino.

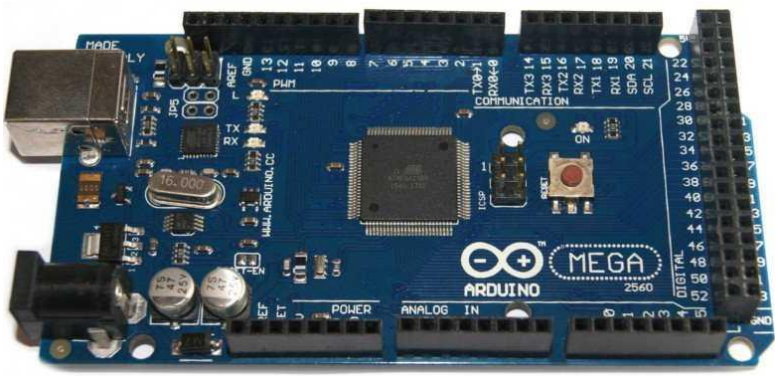

Figura 2.15 Placa Arduino Mega.

El Mega 2560 tiene 16 entradas analógicas, cada una de las cuales proporciona 10 bits de resolución (es decir, 1024 valores diferentes). Por defecto, miden desde tierra hasta 5 volts, aunque es posible cambiar el extremo superior de su rango usando el pin AREF y la función analogReference ().

Hay otro par de pines en la placa:

- AREF. Voltaje de referencia para las entradas analógicas. Usado con analogReference ().
- Reiniciar. Se pone esta línea BAJA para reiniciar el microcontrolador. Normalmente se usa para agregar un botón de reinicio a los escudos que bloquean el de la placa.

(Arduino, 2020)

2.5.2 Arduino Nano

El Arduino Nano es una placa pequeña, completa y compatible con la placa de pruebas basada en el ATmega328. Tiene más o menos la misma funcionalidad del Arduino Duemilanove, pero en un paquete diferente. Solo carece de un conector de alimentación de CC y funciona con un cable USB Mini-B en lugar de uno estándar. La tabla 2.2 muestra algunas características de la placa Arduino Nano.

El Arduino Nano (figura 2.16) cuenta con 14 pines que se pueden utilizar como entrada o salida mediante las funciones pinMode (), digitalWrite () y digitalRead (), que operan a 5 volts. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia pull-up interna (desconectada por defecto) de 20-50 k Ohms.

Tabla 2.2 Datos sobre el Arduino Nano.

Microcontroller	ATmega328
Architecture	AVR
Operating Voltage	5 V
Flash Memory	32 KB of which 2 KB used by bootloader
SRAM	2 KB
Clock Speed	16 MHz
Analog IN Pins	8
EEPROM	1 KB
DC Current per I/O Pins	40 mA (I/O Pins)
Input Voltage	7-12 V
Digital I/O Pins	22 (6 of which are PWM)
PWM Output	6
Power Consumption	19 mA
PCB Size	18 x 45 mm
Weight	7 g

Además, algunos pines tienen funciones especializadas:

- Serial: Pines 0 (RX) y 1 (TX). Se utilizan para recibir (RX) y transmitir (TX) datos en serie TTL. Estos pines están conectados a los pines correspondientes del chip serie FTDI USB a TTL.
- Interrupciones externas: Pines 2 y 3. Estos pines se pueden configurar para disparar una interrupción en un valor bajo, un flanco ascendente o descendente, o un cambio de valor.
- PWM: Pines 3, 5, 6, 9, 10 y 11. Proporcionan una salida PWM de 8 bits con la función analogWrite ().
- SPI: Pines 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Estos pines admiten la comunicación SPI que, aunque proporcionada por el hardware subyacente, no está incluida actualmente en el lenguaje Arduino.
- LED: Pin 13. Hay un LED incorporado conectado al pin digital 13. Cuando el pin es de valor ALTO, el LED está encendido, cuando el pin es BAJO, está apagado.

El Nano tiene 8 entradas analógicas, cada una de las cuales proporciona 10 bits de resolución (es decir, 1024 valores diferentes). Por defecto miden desde tierra hasta

5 voltios, aunque es posible cambiar el extremo superior de su rango usando la función analogReference (). Los pines analógicos 6 y 7 no se pueden utilizar como pines digitales. Además, algunos pines tienen una funcionalidad especializada:

- I2C: Pines A4 (SDA) y A5 (SCL). Admite la comunicación I2C (TWI) mediante la biblioteca Wire (documentación en el sitio web Wiring).

Hay otro par de pines en la placa: AREF:

- Voltaje de referencia para las entradas analógicas. Usado con analogReference ().
- Reiniciar. Se pone esta línea BAJA para reiniciar el microcontrolador. Normalmente se usa para agregar un botón de reinicio a los escudos que bloquean el de la placa.

(Arduino, 2021)

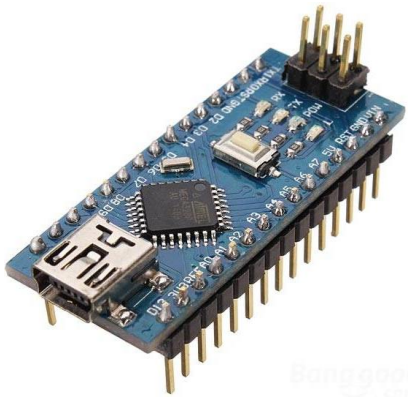

Figura 2.16 Placa Arduino Nano.

2.5.3 Protocolo de comunicación SPI

El bus SPI (Serial Peripheral Interface) fue desarrollado por Motorola en 1980. Sus ventajas respecto a otros sistemas han hecho que se convierta en un *estándar de facto* en el mundo de la electrónica.

El bus SPI tiene una arquitectura del tipo **maestro-esclavo**. El dispositivo maestro puede iniciar la comunicación con uno o varios esclavos, y enviar y recibir datos de ellos. Los dispositivos esclavos no pueden iniciar la comunicación, ni entercambiar datos entre ellos directamente.

En el bus SPI la comunicación de datos entre maestro y esclavos se realiza en dos líneas independientes; una del maestro a los esclavos, y otra de los esclavos al maestro. Por lo tanto, la comunicación se considera *Full Duplex*; es decir, el maestro puede enviar y recibir datos simultáneamente.

Otra característica del SPI es que es un bus síncrono. El dispositivo maestro proporciona una señal de reloj que mantiene a todos los dispositivos sincronizados. Esto reduce la complejidad del sistema frente a los sistemas asíncronos.

Así, el bus SPI requiere de un mínimo de tres líneas de comunicación, denominadas MOSI (Master-Out Slave-In), MISO (Master-In Slave-Out), y SCK (Serial Clock), y una línea adicional denominada SS (Slave Select) para cada dispositivo esclavo conectado (figura 2.17). Esto para seleccionar el dispositivo con el que se va a realizar la comunicación.

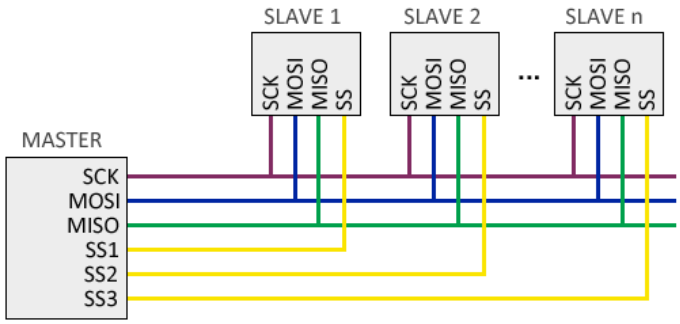

Figura 2.17 Líneas de comunicación SPI.

Por defecto, el maestro mantiene en nivel alto todas las líneas SS. Cuando el maestro quiere establecer comunicación con un esclavo pone a nivel bajo la línea SS correspondiente, lo que indica al esclavo que debe iniciar la comunicación. En cada pulso de la señal de reloj (normalmente en el flanco de subida), el dispositivo maestro envía un bit del esclavo y a la vez recibe in bit del esclavo seleccionado.

Los datos enviados no siguen regla alguna; es decir, se puede enviar cualquier secuencia arbitraria de bits. Esto hace que los dispositivos conectados necesiten tener pre acordado la longitud y significado de lo que van a enviar y recibir. (Llamas, Luis Llamas, 2016)

2.5.4 PWM

PWM son siglas en inglés para ***Pulse Width Modulation***, y se puede traducir como *modulación de ancho de pulso*. La modulación de ancho de pulso está formada por una señal de onda cuadrada que no siempre tiene la misma relación entre el tiempo que está en nivel alto y el tiempo que está en nivel bajo.

En la figura 2.18 se observa una señal que varía entre los 5 Volts y los 0 Volts. A lo largo del tiempo, la señal varía entre estos dos valores de voltaje, pero en un determinado lapso la señal se encuentra en un nivel alto (5 Volts) y durante otro lapso se encuentra en un nivel bajo (0 Volts).

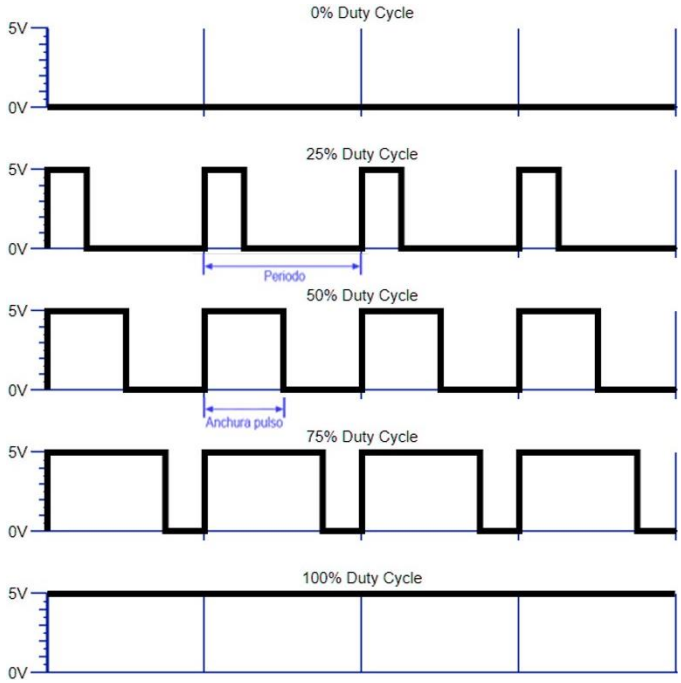

Figura 2.18 Ejemplo de señal PWM.

La suma del tiempo en que la señal se encuentra en nivel alto y en nivel bajo es el periodo de la señal. La señal PWM funciona variando su valor entre dos valores de voltaje establecidos en periodos concretos de tiempo y con una frecuencia fija. Estos periodos reciben el nombre de **ciclo de trabajo** (Duty Cycle, en inglés).

Así, el ciclo de trabajo se define como la relación entre el tiempo en nivel alto y el periodo del PWM. Cuanto mayor sea el ciclo de trabajo, mayor tiempo estará la señal en nivel alto, sin variar el periodo.

Al modificar el ciclo de trabajo de una señal PWM, lo que se hace es variar su tensión media. Cuando una señal de tensión media pasa por ciertos componentes electrónicos, como un LED, motores de corriente continua, ventiladores o zumbadores, puede hacer que su comportamiento cambie. (Gómez, 2010)

2.6 Módulo acelerómetro

Un acelerómetro es un dispositivo con la capacidad de medir y analizar la aceleración linear y angular. Al realizar mediciones de aceleración gravitacional, permite determinar el ángulo de desviación del objeto con respecto al eje vertical. El acelerómetro es un transductor de aceleración que mide su propio movimiento en el espacio.

Los acelerómetros pueden emplearse en estructuras de acero o mástiles, puentes o estructuras de edificios, así como para proteger los discos duros de las computadoras contra daños, en equipos médicos y deportivos, en cámaras y videocámaras, en teléfonos inteligentes, controles remotos, controladores o sistemas de navegación (Grupo TME, 2020).

La aceleración es la variación de la velocidad de un objeto por una unidad de tiempo; es decir, la razón de cambio en la velocidad respecto al tiempo:

$$a = dV/dt \qquad \text{(Ec. 2.1)}$$

De igual forma, la segunda ley de Newton indica que, en un cuerpo que posee su masa constante, la aceleración del cuerpo es proporcional a la fuerza que actúa sobre el mismo cuerpo:

$$a = F/m \qquad \text{(Ec. 2.2)}$$

La velocidad angular es la tasa de cambio del desplazamiento angular por unidad de tiempo; esto es, qué tan rápido gira un cuerpo alrededor de su eje:

$$\omega = d\theta/dt \qquad \text{(Ec. 2.3)}$$

(Naylamp Mechatronics, 2020).

2.6.1 El módulo MPU-6050

El MPU-6050 (figura 2.19) tiene un giroscopio de tres ejes con el que se puede medir la velocidad angular, así como un acelerómetro de tres ejes con el que se

pueden medir los componentes X, Y y Z de la aceleración. Al poder detectar seis mediciones diferentes, se dice que este acelerómetro cuenta con seis grados de libertad (6 Degrees of Freedom). (Naylamp Mechatronics, 2020).

El sensor cuenta con un bus de comunicación I2C, con el que acepta directamente entradas de información de una brújula externa de 3 ejes, para proporcionar una salida de *Motion Fusion* de 9 ejes. Este dispositivo también está diseñado para poder interactuar con múltiples sensores no inerciales digitales, así como sensores de presión, a través de su puerto auxiliar I2C. Además, cuenta con tres convertidores analógico/digital (Analog Digital Converter) de 16 bits para poder digitalizar las salidas del giroscopio, y otros tres convertidores analógico/digital de 16 bits para digitalizar las salidas del acelerómetro.

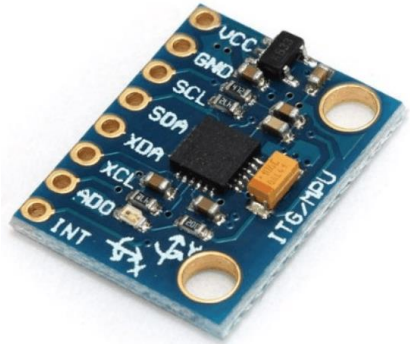

Figura 2.19 Módulo MPU-6050.

Para un seguimiento preciso de movimientos, tanto rápidos como lentos, cuenta con un giroscopio programable por el usuario de escala completa, con rangos de ±250, ±500, ±1000 y ±2000 °/seg (grados por segundo), y un acelerómetro programable por el usuario de escala completa con rangos de ±2g, ±4g, ±8g y ±16g.

Dentro del acelerómetro hay un MEMS (Micro Electro Mechanical System – Sistema Micro Electro-Mecánico), que es el que permite medir la aceleración. La medición es obtenida de una forma similar a como se obtendría en un sistema masa-resorte. Hay que tener en cuenta que, aunque no exista un movimiento, el acelerómetro siempre estará censando la aceleración de la gravedad. Con el MEMS también se puede medir la velocidad angular (InvenSense Inc., 2012).

La dirección de los ejes se muestra en la cara superior del módulo, y esta orientación debe ser totalmente tomada en cuenta para no errar en el signo de las

aceleraciones. Este sensor es utilizado en proyectos de navegación, geometría, estabilización, etc. (Naylamp Mechatronics, 2020)

La distribución de pines, así como la cantidad de estos, es la siguiente:

1.- VCC: Pin de alimentación del módulo. La alimentación no debe ser mayor de 3.5 V.

2.- GND: Tierra (Ground).

3.- SCL: Pin para el bus de entrada de *clock I2C* esclavo.

4.- SDA: Bus de datos *I2C* esclavo.

5.- XDA: Bus maestro de datos *I2C*, para conectar un sensor externo.

6.- XCL: Bus maestro de datos *I2C*, para conectar un sensor externo.

7.- ADO: LSB para la dirección de esclavo en el protocolo *I2C*.

8.- INT: Interrupción digital.

2.7 Módulos de comunicación de Radio Frecuencia

Los módulos de radiofrecuencia, también conocidos como Módulos de RF, son pequeños dispositivos electrónicos que se usan para enviar y recibir señales de radio entre dos dispositivos. Estos módulos suelen utilizarse con un par de codificadores/decodificadores; el codificador se utiliza para codificar datos paralelos para la transmisión, y el decodificador es utilizado para decodificar los datos recibidos.

Los módulos RF facilitan la comunicación inalámbrica, ya que no requieren de una línea de visión. Su rango de funcionamiento suele ser de 3kHz a 300GHz; el rango inferior es usado en submarinos o estaciones de radio, mientras que el rango superior se utiliza para aplicaciones de GPS, Wi-Fi. Bluetooth, y retransmisión de TV.

El rendimiento en aplicaciones donde estos módulos están involucrados depende de algunos factores, tales como el aumentar la potencia del transmisor. Al hacer esto, se puede alcanzar un mayor rango de comunicación, pero también existe una pérdida de energía de la fuente de alimentación, que puede acortar la vida útil de

los dispositivos conectados a dicha fuente. Otro factor a considerar es que, al usar la potencia de transmisión más alta, puede interferir con otras frecuencias de radio. (RS Components, 2020)

2.7.1 El Módulo nRF24L01

El nRF24L01 es un chip de comunicación inalámbrica fabricado por *Nordic Semiconductor*. Integra un transceptor (transmisor + receptor) con un rango de frecuencia de 2.4GHz a 2.5GHz, y una banda libre para uso gratuito. Su velocidad de transmisión es configurada por el usuario, y esta puede ser de 250 kbps (kilo bytes por segundo), 1 Mbps (Mega byte por segundo) o 2 Mbps, y es posible conectar hasta seis dispositivos de forma simultánea.

Existen dos versiones del módulo que montan al nRF24L01. Uno cuenta con una antena integrada en forma de zigzag, y un alcance máximo de 20 a 30 metros; la versión de alta potencia tiene incorporada un amplificador y una antena externa, con un alcance máximo entre 700 y 1000 metros (figura 2.20). Este módulo también cuenta con la lógica necesaria para una comunicación robusta; es decir, corrección de errores o reenvío de datos, en caso de ser necesario, liberando al procesador de hacer esa tarea.

Figura 2.20 Módulo nRF24L01 (izquierda) y Módulo nRF24L01+PA+LNA (derecha).

Su banda de frecuencia va de 2400 MHz a 2525 Hz, lo que hace posible elegir entre 125 canales con un espacio de 1 Mhz (Mega Hertz) de un canal a otro. Es recomendable usar las frecuencias de 2501 a 2525 MHz para evitar interferencias con las redes Wi-Fi (Llamas, 2016).

Su distribución de pines es la siguiente:

1.- GND: Tierra o 0 Volts.

2.- Vcc: Alimentación a 3.3 Vcd.

3.- CE (Chip Eneable): Activa el modo de transmisión o de recepción.

4.- CSN (Chip Select Not): Selecciona al esclavo.

5.- SCK: Señal de reloj para la comunicación SPI.

6.- MOSI (Master Out Slave In): Salida de datos del maestro, entrada de datos al esclavo.

7.- MISO (Master In Slave Out): Entrada de datos del maestro, salida de datos del esclavo.

8.- IRQ (Interrump Request): Pin para interrumpir la transmisión de datos.

(NORDIC SEMICONDUCTOR, 2008)

2.8 Bluetooth

Se denomina Bluetooth al protocolo de comunicaciones diseñado especialmente para dispositivos de bajo consumo, que requieren corto alcance de emisión, y que estén basados en transceptores de bajo costo.

Es una especificación industrial para *redes inalámbricas de área personal* (en inglés, Wireless Personal Area Network – WPAN) que hace posible la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM (Industriales, Científicas y Médicas) de los 2.4 GHz. Estas normas aparecen con la intención de facilitar las comunicaciones entre equipos móviles, eliminar los cables y conectores que estos podrían necesitar, y ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

Los dispositivos Bluetooth (figura 2.21) pueden actuar como “maestros” o “esclavos”. La diferencia entre uno y otro es que un esclavo sólo puede a un maestro y a ningún otro más, mientras que un maestro puede conectarse a varios esclavos o permitir que ellos se conecten, y enviar y recibir información de ello, controlando

la transferencia de información. Pueden conectarse un máximo de siete esclavos a un mismo maestro.

Figura 2.21 Ejemplos de dispositivos que cuentan con Bluetooth.

Cada uno de los dispositivos que se identifican vía Bluetooth presentan una dirección única de 48 bits, además de un nombre del dispositivo que sirve para identificarlo de forma cómoda. Habitualmente, también incluye un PIN de conexión o número de identificación que debe teclearse para obtener acceso al mismo.

Como el Bluetooth fue desarrollado por Nokia para conectar teléfonos móviles a otros dispositivos como auriculares, micrófonos o conexiones al audio del coche, existe un procedimiento definido llamado *Pairing* (emparejamiento), que vincula a dos dispositivos Bluetooth. Cuando se vinculan dos dispositivos Bluetooth, se inicia un proceso en el que ellos se identifican por nombre y dirección interna, y se solicitan la clave PIN para autorizar la conexión. Si el emparejamiento se realiza con éxito, ambos dispositivos suelen guardar la dirección del otro, y cuando se encuentran cerca se vuelven a vincular sin necesidad de intervención manual. (jecrespom, 2021)

2.8.1 El módulo HC-05

Este módulo (figura 2.22) permite una conexión desde Arduino a un celular o PC de forma inalámbrica (Bluetooth), con la facilidad de operación de un puerto serial. La transmisión se realiza en forma transparente al programador, por lo que se conecta en forma directa a los pines seriales del microcontrolador que se esté utilizando

(respetando los niveles de voltaje, ya que el módulo se alimenta con 3.3 Volts). Todos los parámetros del módulo se pueden configurar mediante comandos AT.

Recordando que la comunicación Bluetooth se da entre dos tipos de dispositivos: un maestro y un esclavo, si el objetivo es conectar el módulo a un *smartphone* con sistema operativo *Android*, se puede utilizar el módulo Bluetooth HC-05, que viene configurado de fábrica para trabajar como esclavo; es decir, preparado para escuchar peticiones de conexión, pero también se puede configurar para trabajar como maestro, utilizando comandos AT.

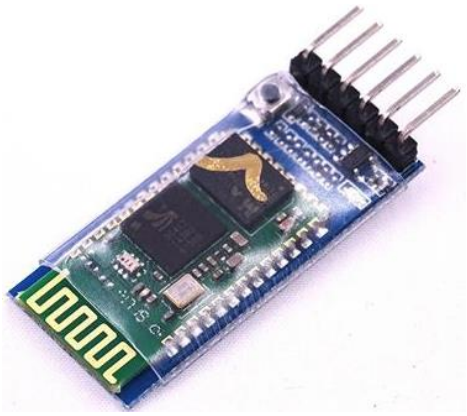

Figura 2.22 Módulo Bluetooth HC05.

A continuación, se muestran algunas características de funcionamiento, así como la distribución de pines.

- Voltaje de operación: 3.6V - 6V DC.
- Consumo corriente: 50mA.
- Bluetooth: V2.0+EDR.
- Frecuencia: Banda ISM 2.4GHz.
- Modulación: GFSK (Gaussian Frequency Shift Keying).
- Potencia de transmisión: 4dBm, Class 2.
- Alcance de 10 metros.
- Interface comunicación: Serial TTL.
- Velocidad de transmisión: Desde 1200bps hasta 1.3Mbps.
- Tasa de baudios por defecto: 38400, 8, 1, n.

(Naylamp Mechatronics, 2021)

Distribución de pines:

1.- STATE: Este pin va conectado a un led que permite visualizar cuándo se comunican datos.

2.- RXD: Recepción de datos.

3.- TXD: Transmisión de datos.

4.- GND: Conexión a tierra.

5.- Vcc: Conexión a fuente de alimentación.

6.- Enable: Este pin debe estar en nivel alto para poner al módulo en modo de configuración.

(Madriaga, 2015)

2.9 Carga inalámbrica

La carga inalámbrica consiste en una pequeña bobina por la que se hace circular una corriente eléctrica, la cual, genera un campo magnético ascendente. Este campo magnético pasa por una segunda bobina, generando una corriente en esta última. La segunda bobina debe estar acomodada de manera que el campo magnético de la primera bobina pueda interactuar con ella, de lo contrario la carga no se efectuará de forma correcta.

La carga inalámbrica es más lenta, y esto se debe a que la transferencia de potencia no es igual. Esto significa que la primera bobina necesita de más potencia para funcionar de la que la segunda bobina puede recibir.

Los dispositivos que cuentan con este tipo de carga suelen tener carcazas o recubrimientos de plástico o cristal, pero no de metal. Esto es porque el metal es un buen conductor de la electricidad. Si se colocara un objeto metálico en un sistema de carga inalámbrica, lo que se cargará será el objeto metálico, provocando que se caliente y podría terminar en un accidente (Roberto CCU, 2017).

2.9.1 Cargador inalámbrico

Para que una persona pueda hacer uso de la carga inalámbrica requiere de un cargador y un dispositivo compatible. Este tipo de carga es mayormente usada para cargar teléfonos celulares, pero también puede ser usada en otros dispositivos.

Para el caso del celular, primero se debe contar con la compatibilidad para carga inalámbrica. Algunos teléfonos modernos cuentan con esa opción por efecto de fábrica, pero para los que no, se tiene la opción de un adaptador. Este adaptador funciona como bobina receptora del campo magnético emitido por la bobina transmisora del cargador (figura 2.23).

Figura 2.23 Cargador inalámbrico (izquierda) y adaptador receptor (derecha) para teléfonos celulares.

Es posible interrumpir la carga con sólo levantar el teléfono de la base del cargador o incluso separar un poco el teléfono de la base y que la carga no se detenga. Como ambas bobinas deben coincidir para un funcionamiento eficaz (figura 2.24), algunos cargadores incorporan varias bobinas para no tener que centrar el teléfono en la base del cargador (Roberto CCU, 2017).

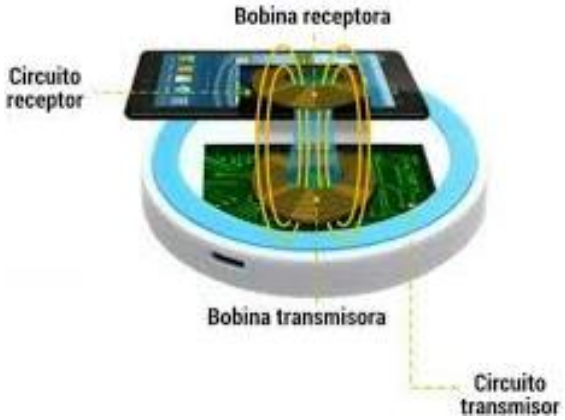

Figura 2.24 Esquema de funcionamiento de la carga inalámbrica en celulares.

2.10 El transistor

Es un dispositivo que regula el flujo de corriente o de voltaje en un circuito, actuando como interruptor o amplificador (o ambos) para señales eléctricas. Este componente está formado por materiales semiconductores, y pueden ser dos del tipo P y uno del tipo N (transistores PNP) o dos del tipo N y uno del tipo P (transistores NPN).

Los transistores cuentan con tres terminales: Base, colector y emisor. Es importante identificar de forma correcta estas terminales antes de conectarlo a un circuito, pues varían de lugar dependiendo del tipo de transistor (figura 2.25).

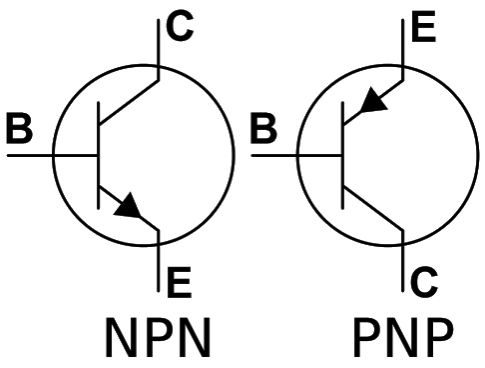

Figura 2.25 Símbolo eléctrico del transistor.

Su funcionamiento se resume a partir de la corriente en la base. Si no hay corriente en la base, no habrá corriente entre el colector y el emisor; cuando llega una pequeña corriente por la base, existirá una corriente entre el colector y el emisor, siendo esta corriente mayor a la de la base. A partir de este funcionamiento, el transistor puede ser utilizado como un oscilador, conmutador, rectificador de corriente eléctrica, interruptor o amplificador. (Area Tecnología, 2020)

2.10.1 MOSFET

Es un tipo de transistor utilizado para la conmutación y amplificación de señales eléctricas. Su nombre completo es *Transistor de Efecto de Campo de Metal-óxido-semiconductor* (en inglés, Metal Oxide Semiconductor Field Effect Transistor) y se debe a la construcción del mismo. Como se ve en la figura 2.26, los MOSFET también cuentan con tres terminales, y se dividen en tipo canal N y tipo canal P.

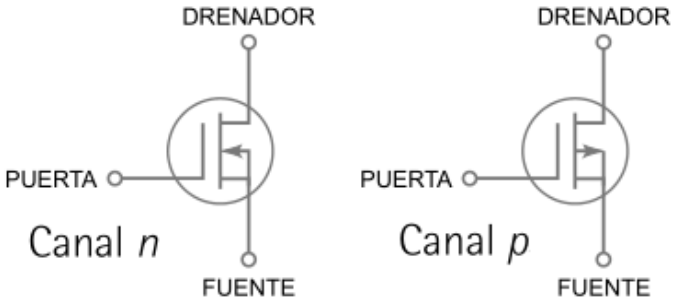

Figura 2.26 Símbolo eléctrico del MOSFET.

Estos dispositivos son usados para la conmutación de señales de alta velocidad, así como para el control digital de cargas de mayor voltaje y mayor corriente a los valores nominales que puede soportar un microcontrolador. (García Gonzáles, 2016)

2.10.2 MOSFET irf640N

El MOSFET de potencia HEXFET de quinta generación, del fabricante ***International Rectifier*** utiliza técnicas de procesamiento avanzadas para lograr una resistencia de encendido extremadamente baja por área de silicio. Este beneficio, combinado con la velocidad de conmutación rápida y el diseño de dispositivo robusto por el que son bien conocidos los MOSFET de potencia HEXFET, proporciona al diseñador un dispositivo extremadamente eficiente y confiable para su uso en una amplia variedad de aplicaciones.

El encapsulado TO-220 es preferido para aplicaciones comerciales-industriales con niveles de disipación de energía de aproximadamente 50 W. La baja resistencia térmica y el bajo costo de empaque del TO-220 contribuyen a su amplia aceptación en la industria. El *D2Pak* es un paquete de energía de montaje en superficie capaz de acomodar tamaños de troquel hasta HEX-4. Proporciona la capacidad de potencia más alta y la resistencia de encendido más baja posible en cualquier paquete de montaje en superficie existente. El *D2Pak* es adecuado para aplicaciones de alta corriente debido a su baja resistencia de conexión interna y puede disipar hasta 2 W en una aplicación típica de montaje en superficie.

A continuación, se mencionan algunas características de este dispositivo, así como su distribución de pines.

- MOSFET tipo Canal N.

- Conmutación rápida.
- Requisitos de accionamiento simples.

Su distribución de pines aparece en la figura 2.27:

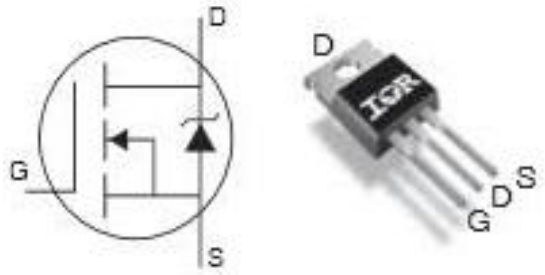

Figura 2.27 Diagrama de pines del MOSFET irf640N.

En donde:

1.- *Gate* o Puerta (G): Pin por donde se recibe la señal que activa al MOSFET, permitiendo un paso de corriente de *Source* (S) a *Drain*(D).

2.- *Drain* o Drenador (D): Pin conectado a una fuente de poder de donde se alimentará a la carga que se desea controlar.

3.- *Source* o Fuente (S): Pin conectado a la carga que se desea controlar.

(International Rectifier, 2021)

2.11 App Inventor

MIT App Inventor es un entorno de programación visual e intuitivo (figura 2.28) que permite crear aplicaciones completamente funcionales para teléfonos inteligentes y tabletas. Aún si se es nuevo en este entorno, se puede obtener una aplicación simple y funcional en menos de treinta minutos, además de que cuenta con una herramienta basada en bloques que facilita la creación de aplicaciones complejas y de alto impacto en un tiempo menor que otros entornos de programación.

Un pequeño equipo de personas y estudiantes de CSAIL (Computer Science and Artificial Intelligence Laboratory – Laboratorio de Informática e Inteligencia Artificial), dirigido por el profesor Hal Abelson, forman el núcleo de un movimiento internacional de inventores. Además de liderar el alcance educativo en torno a App Inventor y realizar investigaciones sobre sus impactos, este equipo central mantiene

el entorno de desarrollo de aplicaciones en línea gratuito que sirve a más de 6 millones de usuarios registrados.

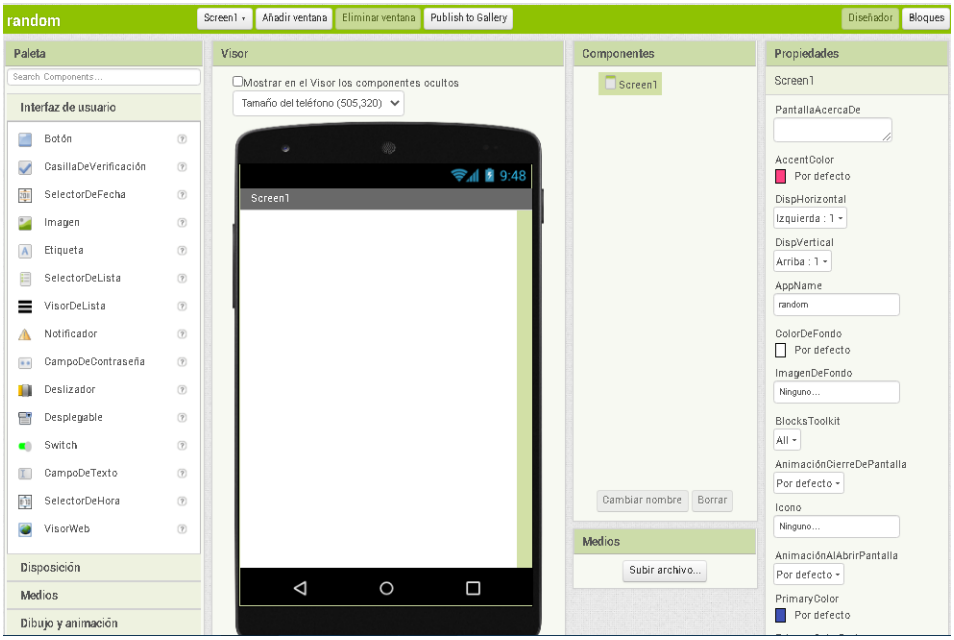

Figura 2.28 Entorno para desarrollo de aplicaciones de App Inventor.

Capítulo 3 Desarrollo experimental

Como se mencionó en la introducción, este trabajo parte del realizado por (Mateos Peña, 2018), el cual, controla la iluminación de unas tiras led RGB presentes en un módulo maestro a través de la orientación de varios módulos esclavos. La información de la posición en los esclavos es enviada mediante radiofrecuencia, y el maestro recoge la información y la utiliza para cambiar los colores en las tiras led.

Para la realización de este proyecto se plantearon diferentes objetivos de acuerdo a las nuevas necesidades de los módulos maestro y esclavos. Para los esclavos se pensó en adicionar un zumbador para emitir un sonido en cada cambio de posición, así como un medidor de la carga existente en la batería que se utiliza para alimentar al mismo módulo, la cual, se cargará mediante un módulo inalámbrico. También, se cambió el dispositivo de radiofrecuencia que utilizó (Mateos Peña, 2018) por uno de mayor rango de transmisión, y se adecuó un regulador de voltaje fijo en 3.3 volts para optimizar el funcionamiento del mismo.

Adicionalmente, se creó una App para dispositivos móviles Android, la cual permite observar el nivel de carga de las baterías, y activar-desactivar los zumbadores.

Para el maestro se hizo la integración de una etapa de potencia de mayor capacidad, dado que éste prototipo está pensado que pueda trabajar con tiras LED de tres metros de largo, lo que aumenta la necesidad de potencia, además de un módulo Bluetooth con la que se pueda hacer comunicación entre el maestro y la App.

Para llevar a cabo lo anteriormente propuesto, se definieron cuatro etapas de desarrollo:

1.- Diseño electrónico de acuerdo a los nuevos requerimientos.

2.- Programación de algoritmos de control.

3.- Desarrollo de la aplicación.

4.- Pruebas de funcionamiento.

3.1 Diagrama de bloques

El siguiente diagrama de bloques (figura 3.1) ilustra, de forma general, el funcionamiento de los módulos maestro y esclavos en conjunto, además de la aplicación, la cual se encuentra operando desde un teléfono móvil. Más adelante se presentará a detalle el funcionamiento de cada módulo, así como de los dispositivos que lo componen.

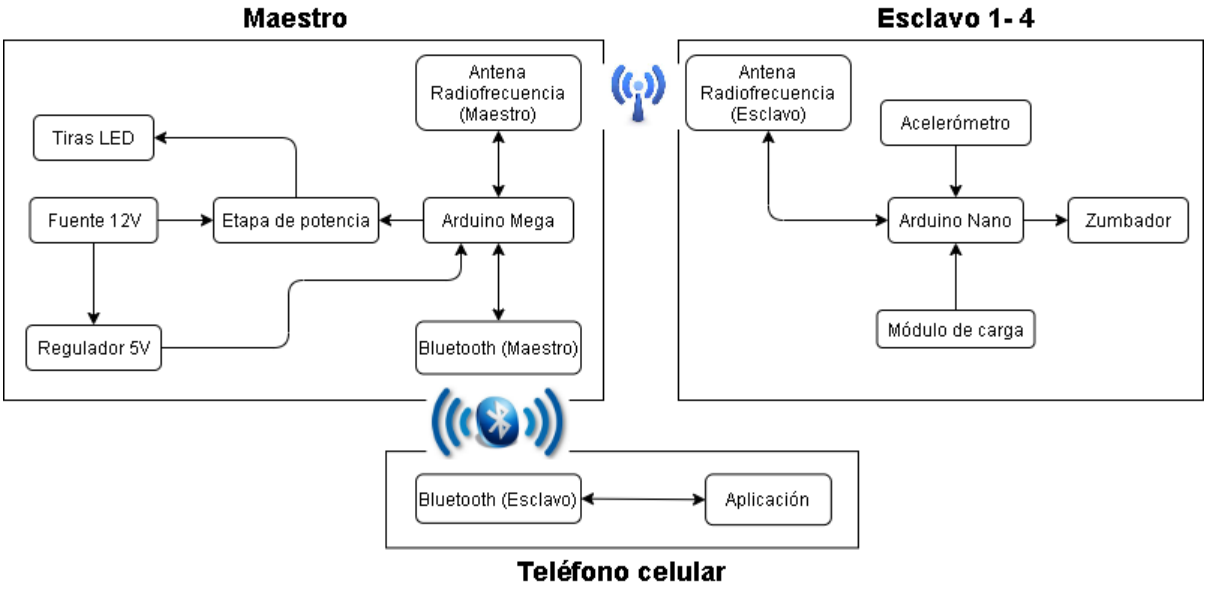

Figura 3.1 Diagrama de bloques del funcionamiento del prototipo.

Donde:

- **Maestro**: Módulo con el rol de maestro, en el que se encuentra el Arduino Mega, la fuente de alimentación, la iluminación de las tiras LED, la etapa de potencia, la antena de radiofrecuencia y la antena Bluetooth.
- **Arduino Mega**: Microcontrolador utilizado para interpretar la información obtenida a través de las antenas radiofrecuencia y Bluetooth. Esta información es utilizada para controlar aspectos del módulo, como la etapa de potencia.
- **Antena Radiofrecuencia (Maestro)**: Dispositivo utilizado para la comunicación bidireccional entre el módulo maestro y los módulos esclavos.

- **Bluetooth (Maestro)**: Dispositivo utilizado para la comunicación bidireccional entre el módulo maestro y el teléfono celular en el que se encuentre operando la aplicación.
- **Etapa de potencia**: Arreglo de circuitería necesario para suministrar la potencia requerida por las tiras LED.
- **Tiras LED**: Tiras LED que prenderán de un color determinado. Dicho color es asignado mediante la codificación presente en el Arduino Nano de los módulos esclavos, e interpretado y accionado por el Arduino Mega del módulo maestro.
- **Fuente 12V**: Fuente de poder que suministra voltaje a las tiras LED, así como suministra de energía al Arduino Mega.
- **Esclavo 1-4**: Módulos con el rol de esclavo, en el que se encuentra el Arduino Nano, el acelerómetro, el zumbador, la antena radiofrecuencia, y el módulo de carga de la batería que alimenta al esclavo (uno por cada esclavo). Se hace la conjunción de “1-4” porque los cuatro módulos esclavos cuentan con la misma estructura interna.
- **Arduino Nano**: Microcontrolador utilizado para cuantificar la orientación del módulo esclavo y el nivel de voltaje con el que cuenta la batería que lo alimenta, así como asignar un código específico para dicha información cuantificada y hacer que la antena de radiofrecuencia envíe esta información. También, puede cuantificar la información que obtenga la antena de radiofrecuencia y, con esto, controlar aspectos del módulo, como activar o desactivar al zumbador.
- **Antena Radiofrecuencia (Esclavo):** Dispositivo utilizado para la comunicación bidireccional entre el módulo esclavo y el módulo maestro.
- **Acelerómetro:** Dispositivo utilizado para obtener la orientación del módulo esclavo con respecto del suelo.
- **Zumbador:** Dispositivo utilizado para emitir sonido.
- **Módulo de carga:** Módulo en el que se encuentra la batería y un dispositivo receptor de un cargador inalámbrico para recargar la batería.
- **Teléfono celular:** Dispositivo Android en el que se hará uso de la aplicación.

- **Aplicación**: Interfaz desarrollada con el propósito de realizar las configuraciones elegidas por el usuario para los módulos esclavos, como lo es el encendido y apagado del zumbador, o revisar el nivel de batería con el que cuenta cada módulo esclavo.

3.2 Módulo Maestro

Como ya se mencionó anteriormente, el módulo con el rol de maestro está integrado por varios dispositivos, como lo son la antena radiofrecuencia, la antena Bluetooth, y la fuente de alimentación, con los cuales se pueden llevar a cabo los objetivos propuestos en este proyecto. En la figura 3.2 se puede apreciar un diagrama de flujo que ilustra el algoritmo del módulo maestro.

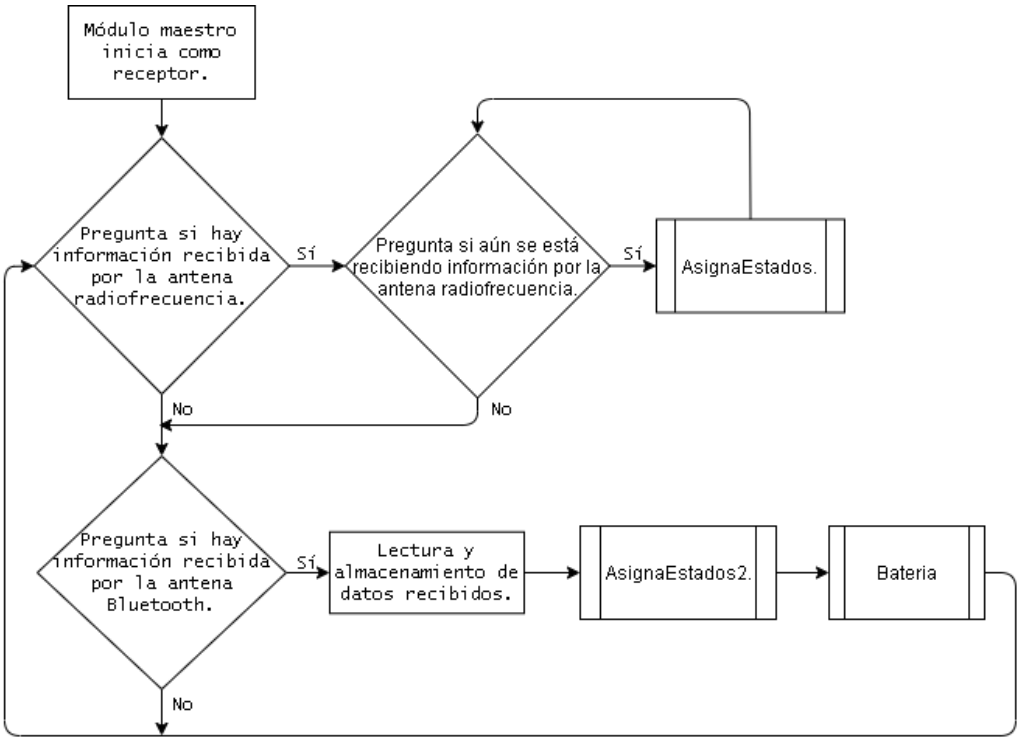

Figura 3.2 Diagrama de flujo del módulo maestro.

En la figura anterior se puede ver que, dentro del diagrama, hay bloques que indican un salto a una subrutina. Más adelante se hará una explicación de estas subrutinas.

Hay que hacer mención importante en que la antena radiofrecuencia con la que cuenta el módulo maestro debe iniciar funcionando como receptor. Esto debido a que el módulo maestro estará a la expectativa de información por parte de alguno

de los cuatro esclavos, pues es mayor el flujo de información recibida, que el flujo de la información que se envía. Una vez recibida la información por parte de la antena radiofrecuencia, realiza un salto a una subrutina llamada “asignaEstados”, en la que se asigna el color en el que las tiras LED deben prender, así como el nivel de voltaje en la batería del esclavo que envió de información.

Si ya no hay datos recibidos en la antena radiofrecuencia, lo siguiente es verificar si hay recepción de datos por la antena Bluetooth. En caso de existir datos, se leen y se almacenan, para luego ser usados por alguna de las dos subrutinas “asignaEstados2” o “Bateria”, según sea necesario.

3.2.1 Programación en Arduino

La programación inicia con la declaración de las bibliotecas que se van a utilizar. Estas bibliotecas son “SPI”, para el protocolo de comunicación SPI, así como “nRF24L01” y “RF24”, utilizadas para poder trabajar con la antena radiofrecuencia. La declaración de estas bibliotecas quedó de la siguiente manera:

```
#include <SPI.h>  //Biblioteca para comunicarse con el nRF24.
#include <nRF24L01.h>
#include <RF24.h> //Bibliotecas para el nRF24.
```

A continuación, se definieron las variables globales del programa, posteriormente se codificó la función *setup*, en la que se inicia al puerto serial con una velocidad de 9600 baudios. También, se inicia a la antena radiofrecuencia, con sus respectivas características, como lo son el nivel de transmisión, los canales de lectura, y el rol de la antena, siendo este el de receptor. Igualmente, se configuran los pines en los que van conectadas las salidas para las señales que harán prender a las tiras LED según sea el caso, así como una salida extra para un LED que servirá como comprobación visual de que el maestro ha conseguido enviar información hacia alguno de los módulos esclavos. Una porción de este código es la siguiente:

```
void setup()
{
  Serial.begin(9600); //Se inicia el puerto serial.
  radio.begin();  //Inicio del módulo RF.
```

```
 radio.setRetries(15, 15); //Configuración del número máximo de reintentos.
 radio.setPALevel(RF24_PA_MIN);  //Nivel de transmisión. Se usa el nivel mínimo para pruebas.
 radio.setDataRate(RF24_250KBPS);  //Se define la velocidad de envío de datos.
 radio.setChannel(108);  //Se fija un canal de comunicación para disminuir la interferencia.
 radio.openReadingPipe(0, dRadio[0]);
 radio.openReadingPipe(1, dRadio[1]);
 radio.openReadingPipe(2, dRadio[2]);
 radio.openReadingPipe(3, dRadio[3]);  //Canales de lectura.
 radio.startListening();//Se incia al maestro como Receptor.
 //Salidas PMW para activar las tiras led.
 pinMode(2, OUTPUT); // 1X
          .
          .
          .
 pinMode(13, OUTPUT); // 4Z
 pinMode(Led, OUTPUT); //Led de prueba.
}
```

Después, se realizó la programación de la función *loop*, en la cual se evalúa si la antena radiofrecuencia ha recibido información. En caso de haber, guarda esa información para que pueda ser usada después por la subrutina “asignaEstados”. Si ya no hay datos recibidos en la antena radiofrecuencia, se pasa a evaluar si hay datos existentes en el monitor serial, pues es aquí en donde está conectada la antena Bluetooth. En caso de haberlos, también se almacenan para luego ser usados por las subrutinas “asignaEstados2” o “Batería”. Este código se muestra a continuación:

```
void loop()
{
 if(radio.available()) //Verificando si hay recepción de datos.
 {
  while(radio.available())
  {
   radio.read(&DatoR, sizeof(DatoR));  //Lee el dato recibido y lo guarda en DatoR.
   //Serial.println(DatoR);
   asignaEstados(DatoR);  //Subrutina de encendido de las tiras led.
  }
```

```
 }
 else
 {
        if (Serial.available()>0) //Si hay datos disponibles en el puerto serie bluetooth, los lee y los guarda
        {                         //en la variable 'dato' para poder usarlo según sea el caso.
                Dato = Serial.read(); //Guardando la información del puerto serial en la variable 'Dato'.


                asignaEstados2(Dato);  //Subrutina para encendido-apagado del zumbador presente en los
        esclavos.


                Bateria(Dato);  //Subrutina para envíar a la app el nivel de batería que tienen los esclavos.
                delay(50);
        }
 }
}
```

3.2.1.1 Subrutinas

En el siguiente apartado se hará la explicación de las subrutinas generadas. Esta explicación viene acompañada por un diagrama de flujo sobre el funcionamiento de cada subrutina.

La primera subrutina invocada en el programa principal es la llamada “asignaEstados”. En esta subrutina se lleva a cabo la codificación para prender las tiras LED, así como asignar el nivel de batería para los módulos esclavos. En la figura 3.3 se muestra el diagrama de flujo de cómo funciona esta subrutina.

Para llevar a cabo esta asignación de código, se utiliza estructura de control *switch-case*, la cual, compara el valor del dato obtenido de forma previa por la antena radiofrecuencia, contra una serie de valores establecidos. El hecho de que haya 96 datos con los que se pueda hacer una comparación (casos que se pueden cumplir) es porque el código para el nivel de batería se asigna en conjunto con el color que debe mostrar la tira LED que le corresponda al esclavo.

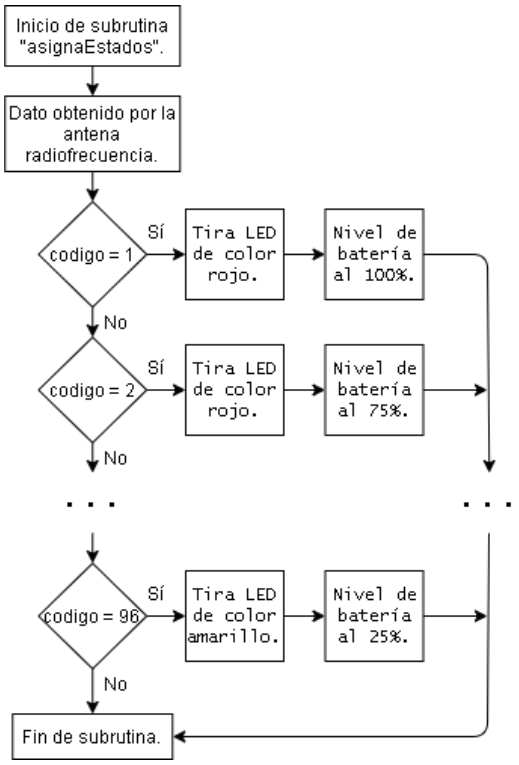

Figura 3.3 Diagrama de flujo de la subrutina "asignaEstados".

Es decir, cuando la primera tira LED, que es la que corresponde al primer esclavo, prende en color rojo, se asigna también el nivel de batería presente en ese esclavo. Como se está considerando el nivel de batería en cuatro estados, que son 100, 75, 50, y 25 por ciento, al color rojo le corresponden los primeros cuatro casos del *switch-case*. Lo mismo aplica para el resto de colores, siendo que los casos correspondientes se dan de forma consecutiva: al verde, del 5 al 8; al azul, del 9 al 12; al cian, del 13 al 16, al magenta, del 17 al 20; al amarillo, del 21 al 24. Con esto, los primeros 24 casos del *switch-case* corresponden a la primer tira LED, los siguientes 24 casos corresponden a la segunda tira LED, y así sucesivamente para las otras dos tiras LED. De aquí que se obtengan 96 casos para el *swtich-case*. A continuación, se observa un fragmento del código que corresponde a la subrutina “asginaEstados”, en la que se encuentra la estructura *switch-case* con la que se asigna el color en las tiras LED y el nivel de batería en los esclavos:

```
void asignaEstados(int codigo)
{
  switch(codigo)  //Selección e impresión de colores.
```

```
 {
  //Esclavo #1
  case 1:  //Rojo
  analogWrite(3,0);
  analogWrite(4,0);
  analogWrite(2,255);
  a = 11;
  break;

  case 2:  //Rojo
  analogWrite(3,0);
  analogWrite(4,0);
  analogWrite(2,255);
  a = 17;
  break;
  .
  .
  .
  case 96:
  analogWrite(46,0);
  analogWrite(44,255);
  analogWrite(45,255);
  d = 12;
  break;
 }
}
```

La siguiente subrutina invocada en el código principal es la llamada “asignaEstados2”. En esta subrutina se lleva a cabo el envío de información hacia los esclavos para prender o apagar los zumbadores que se encuentran en ellos, recordando que esta petición es hecha desde la aplicación desarrollada para dispositivos Android. En la figura 3.4 se muestra el diagrama de flujo del funcionamiento de esta subrutina.

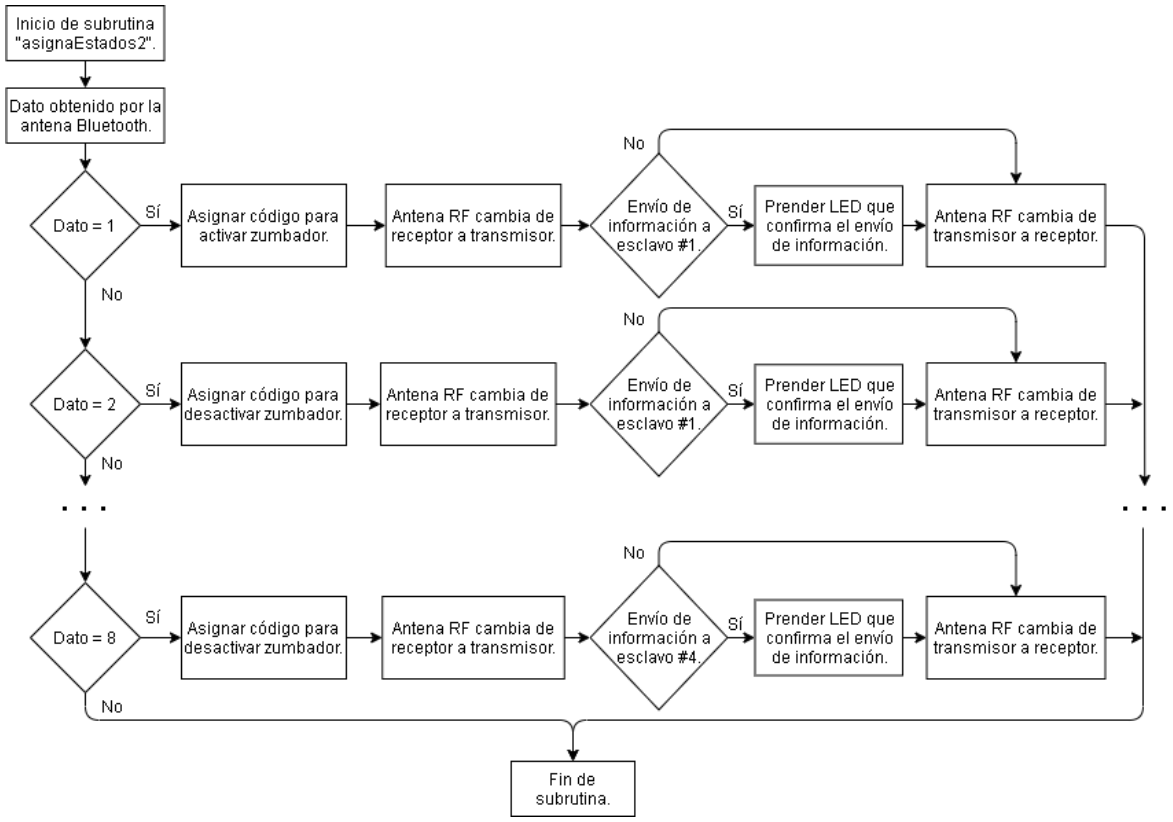

Figura 3.4 Diagrama de flujo de la subrutina "asignaEstados2".

Al iniciar la subrutina, compara el valor del dato previamente obtenido por la antena Bluetooth contra un valor establecido en el programa. Si el valor del dato obtenido por la antena Bluetooth coincide con alguno de los valores establecidos, se realiza la tarea asignada en ese valor. Existen ocho valores con los que se compara el dato obtenido por la antena Bluetooth. Esto es porque cada módulo esclavo tiene dos estados para los zumbadores: encendido y apagado. Al haber dos estados por cada módulo esclavo, y al ser cuatro módulos esclavos, se obtienen ocho valores para cada estado posible de los módulos esclavos. Así, los primeros dos valores corresponden al primer esclavo, los siguientes dos al segundo esclavo, y así sucesivamente para el tercer y cuarto esclavo. Se muestra una fracción del código con el que se lleva a cabo lo anteriormente comentado:

```
void asignaEstados2()
{
  if(Dato == '1')  //Condiciones para prender o apagar los buzzer.
    {
      byte val = 1;  //Dato a envíar.
```

```
  radio.stopListening();  //Maestro pasa a modo de transmisor.
  radio.openWritingPipe(dRadio[0]); //Dirección para el módulo transmisor #1.
  if(radio.write(&val, 1))  //Verificación del envío del dato.
  {
    digitalWrite(Led, HIGH);
    delay(500);
    digitalWrite(Led, LOW); //Condición para comprobar el envío del dato.
  }
  radio.startListening(); //Maestro regresa a modo de receptor.
}
else
  if(Dato == '2')
  {
    byte val = 2;  //Dato a envíar.
    radio.stopListening();  //Maestro pasa a modo de transmisor.
    radio.openWritingPipe(dRadio[0]); //Dirección para el módulo transmisor #1.
    if(radio.write(&val, 1))  //Verificación del envío del dato.
    {
      digitalWrite(Led, HIGH);
      delay(500);
      digitalWrite(Led, LOW); //Condición para comprobar el envío del dato.
    }
    radio.startListening();
  }
  .
  .
  .
        else
         if(Dato == '8')
         {
           byte val = 2;  //Dato a envíar.
           radio.stopListening();  //Maestro pasa a modo de transmisor.
           radio.openWritingPipe(dRadio[3]); //Dirección para el módulo transmisor #4.
           if(radio.write(&val, 1))  //Verificación del envío del dato.
           {
             digitalWrite(Led, HIGH);
             delay(500);
```

```
            digitalWrite(Led, LOW); //Condición para comprobar el envío del dato.
          }
          radio.startListening(); //Maestro regresa a modo de receptor.
        }
}
```

Al iniciar la subrutina, se guarda el dato obtenido por la antena Bluetooth como un *char*, para poder ser comparado con los otros valores del mismo tipo. Luego, vienen una serie de condicionales *if*, en los que se compara, uno por uno, al valor *char* guardado contra los *char* establecidos en el código. Al coincidir con alguno, se genera una variable llamada “val”, de tipo *byte*, y se le asigna el valor de ‘1’ o ‘2’ para indicar si el zumbador debe estar prendido o apagado, respectivamente. Luego, se cambia el rol de la antena radiofrecuencia, de receptor a transmisor, y se abre el canal de comunicación con el módulo esclavo correspondiente. Si el envío de información hacia el esclavo es correcto, se prende el LED auxiliar por un tiempo de 500 ms, que confirma que el proceso de envío fue hecho de manera correcta. Por último, se cambia nuevamente el rol de la antena radiofrecuencia, de transmisor a receptor.

La última subrutina invocada en el programa principal es la llamada “Batería”. En esta subrutina, se asigna el valor de voltaje presente en la batería de los módulos esclavos, en términos de porcentaje. En la figura 3.5 se muestra la forma en que opera esta subrutina.

Al entrar a esta subrutina, se compara el valor del dato obtenido por la antena Bluetooth contra una serie de valores establecidos. En el momento en que el valor obtenido y el valor establecido coinciden, se envía a la aplicación el nivel de batería del esclavo correspondiente.

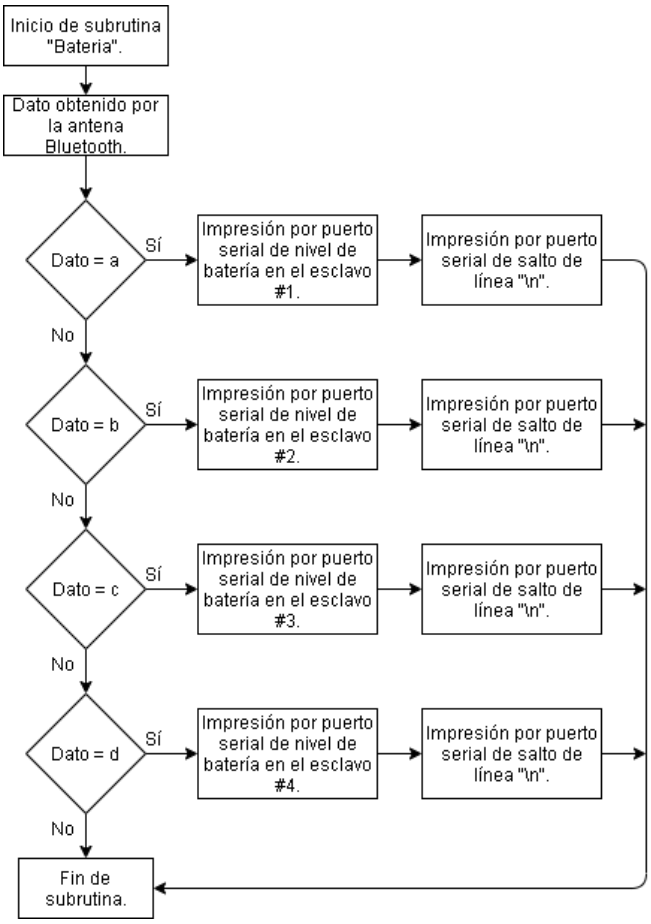

Figura 3.5 Diagrama de flujo de la subrutina "Bateria".

A continuación, se muestra el código que permite llevar a cabo esta tarea:

```
void Bateria()
{
  if (Dato == 'a')  //Condición para envíar el nivel de batería del esclavo #1.
  {
    Serial.print(a);
    Serial.print("\n");
  }
  else{
    if(Dato == 'b') //Condición para envíar el nivel de batería del esclavo #2.
    {
      Serial.print(b);
      Serial.print("\n");
    }
    else{
```

```
    if(Dato == 'c') //Condición para envíar el nivel de batería del esclavo #3.
    {
      Serial.print(c);
      Serial.print("\n");
    }
    else{
      if(Dato == 'd') //Condición para envíar el nivel de batería del esclavo #4.
      {
        Serial.print(d);
        Serial.print("\n");
      }
    }
   }
  }
}
```

Como se puede observar, una vez se salta a esta subrutina, el dato obtenido por la antena Bluetooth se guarda en una variable llamada “dato”, de tipo *char*. Luego, se entra a una serie de condicionales *if*, en los que se compara al *char* guardado al inicio de la subrutina contra los *char* establecidos en el código. Cuando se cumple algún condicional, se mostrará en el monitor serial a la variable que contiene el valor para el nivel de voltaje de la batería de uno de los esclavos, a través de la función *Serial.print*.

3.2.2 Diagrama esquemático y de Tarjeta de Circuito Impreso (PCB)

La figura 3.6 muestra el diagrama esquemático en donde se contemplan las conexiones entre la tarjeta Arduino y los dispositivos nRF24L01 (antena radiofrecuencia), HC-05 (antena Bluetooth), la fuente de alimentación, la etapa de potencia (transistores) y las terminales (borneras) para las tiras LED.

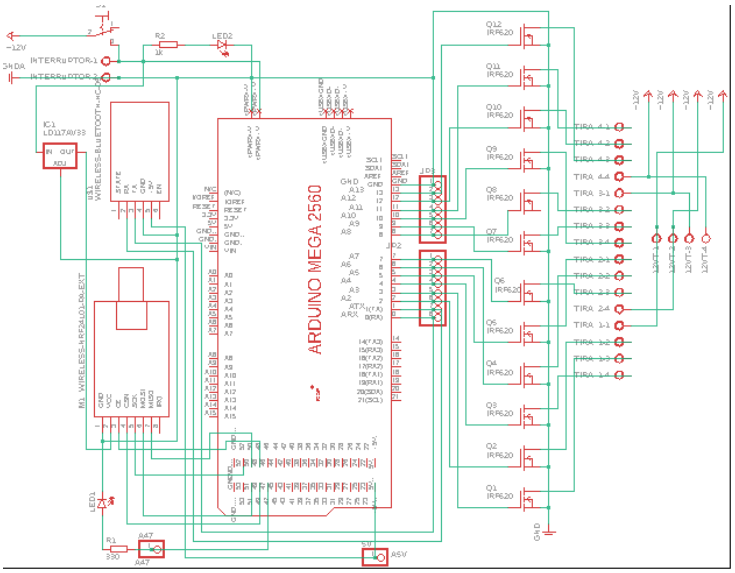

Figura 3.6 Diagrama esquemático del módulo maestro.

A partir de este diagrama esquemático, se elaboró el modelo PCB para la placa fenólica, que se muestra en la figura 3.7.

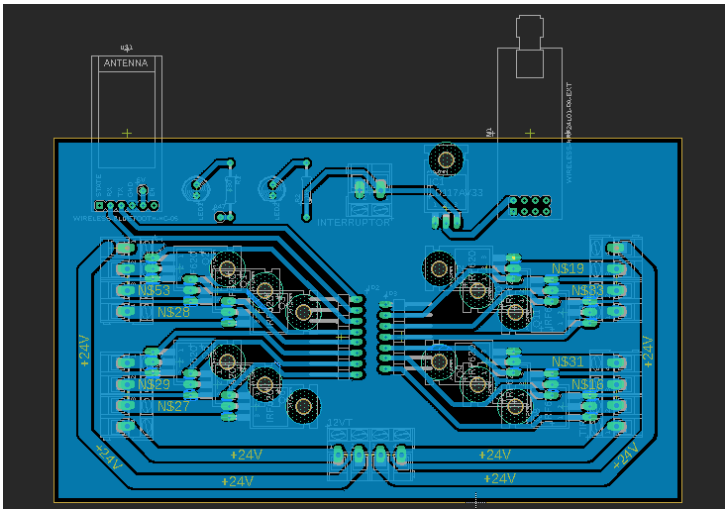

Figura 3.7 Diagrama PCB del módulo maestro.

3.3 Módulos Esclavos

Así como el módulo maestro, los módulos con el rol de esclavo están integrados por varios dispositivos para su funcionamiento, como lo son la antena radiofrecuencia, el acelerómetro, y el zumbador. La figura 3.8 muestra un diagrama de flujo con el que se presenta el funcionamiento de los módulos esclavos. Sólo se muestra el de un módulo esclavo, puesto que los otros tres cuentan con las mismas

características. La parte de la programación difiere un poco entre los módulos esclavos, y esto será mencionado a detalle en este apartado.

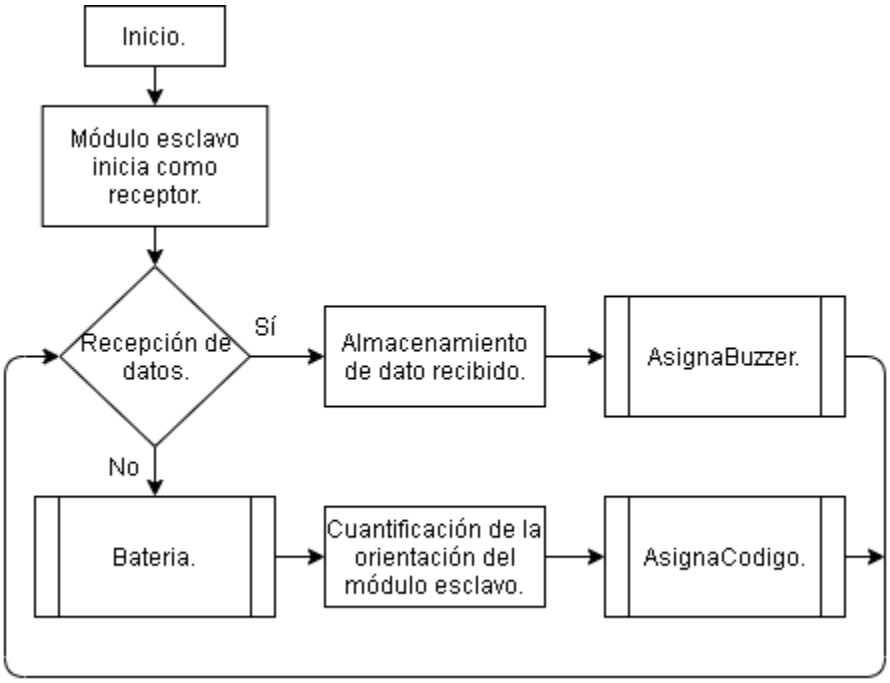

Figura 3.8 Diagrama de flujo de los módulos esclavos.

Pese a que los módulos esclavos envían información al módulo maestro, se debe hacerlos permanecer como receptores, y sólo cambiar al rol de transmisor cuando se deba enviar información al maestro. Esto debido a que las antenas de radiofrecuencia no pueden llevar a cabo una comunicación *Full-duplex*, por lo que deben detener por completo la recepción de datos para transmitir, y detener la transmisión de datos para recibir. Al igual que el módulo maestro, los módulos esclavos cuentan con subrutinas, que se explicarán más adelante a detalle.

3.3.1 Programación en Arduino

Este código inicia declarando las bibliotecas “SPI”, “nRF24L01”, “RF24” y “MPU6050”, las cuales sirven para poder trabajar con el protocolo de comunicación SPI, la antena radiofrecuencia y el acelerómetro:

```
#include <SPI.h>  //Biblioteca para comunicarsecon el RF24.
#include <nRF24L01.h>
#include <RF24.h> //Bibliotecas para el RF24.
#include <MPU6050.h>  //Biblioteca para el MPU6050.
```

Lo siguiente es programar la función *setup*, en donde se configura al módulo serial con una velocidad de 9600 baudios, así como las características de la antena radiofrecuencia, como lo son el nivel de transmisión, el canal de transmisión, y su rol como receptor. También, se configura la salida que accionará al zumbador, y se inicializan las variables para las lecturas del acelerómetro, a la par que se configura una comprobación auditiva (con ayuda del mismo zumbador) de que el acelerómetro se encuentra operando de forma correcta. Este es el código obtenido:

```
void setup()
{
  Serial.begin(9600);//Se incia el puerto Serial.
  radio.begin();//Inicio del módulo RF.
  radio.setRetries(15, 15);//Configuración del número máximo de intentos
  radio.setPALevel(RF24_PA_MIN);  //Nivel de transmisión. Se usa el nivel mínimo para pruebas.
  radio.setDataRate(RF24_250KBPS);//Se define la velocidad de envío de datos.
  radio.setChannel(108);//se fija un canal de comunicación para disminuir la interferencia.
  radio.openReadingPipe(0,PTXpipe);//Canal de lectura.
  radio.startListening();//Se inicia la escucha de datos. Se incia al esclavo como Receptor.

  pinMode(buz,OUTPUT); //Declarando salida para el zumbador.

  for (int lecturaA = 0; lecturaA < numLecturas; lecturaA++)//Se inician todas las lecturas a la entrada en cero.
    lecturas[lecturaA] = 0;
  for (int lecturaB = 0; lecturaB < numLecturas2; lecturaB++)
    lecturas2[lecturaB] = 0;
  for (int lecturaC = 0; lecturaC < numLecturas3; lecturaC++)
    lecturas3[lecturaC] = 0;
  mpu.initialize(); //Se inicia al acelerómetro.
  if (!mpu.testConnection())//Se comprueba que hay comunicación con el acelerómetro.
  {
    tone(buz, 600, 250);
  }
}
```

En la función *loop*, lo primero es una estructura de control *if*, en la que se verifica que haya un dato recibido por parte de la antena radiofrecuencia. En caso de haberlo, almacena el valor de este dato para después ser usado por la subrutina

“asignaBuzzer”. De no haber dato recibido, pasa a la subrutina “Bateria”, y de regreso en el programa principal, realiza la cuantificación de la orientación del módulo esclavo, con respecto del suelo, mediante una serie de operaciones, para después simplificar la información cuantificada de los componentes X, Y, Z con los que se obtiene la orientación y se hace un escalamiento de estos. Este escalamiento es para determinar los grados con los que está inclinado el módulo esclavo. Esta información escalada, junto con el valor de un dato adicional obtenido de la subrutina “Bateria”, son utilizados por una subrutina llamada “asignaCodigo”, siendo su invocación la última en el código principal. A continuación, se muestra el código de la función *loop*:

```
void loop()
{
 if(radio.available()) //Comprobación de datos entrantes en el módulo Rf.
 {
  radio.read(&DatoR, 1);  //Guardando los datos recibidos.

  asignaBuzzer(DatoR); //Subrutina de encendido-apagado del zumbador.
 }
 else
 {
  Bateria();
  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz); //Se obtienen variables del sensor.
  ax = -(ax / 1000);  //Mapear monitor adelante-atrás.
  ay = (ay / 1000); //Mapear monitor izquierda-derecha.
  az = (az / 1000); //Mapear monitor arriba-abajo.
 //************************* Filtro eje X *************************//
  total = total - lecturas[indice]; //Se resta la última lectura.
  lecturas[indice] = ax; //Lecturas del sensor.
  total = total + lecturas[indice]; //Se añade la lectura al total.
  indice = indice + 1;  //Se avanza a la próxima posición del array.
  //Cuando se alcanza el final del array.
  if (indice >= numLecturas) indice = 0; //Se vuelve al inicio.
  //Se calcula el promedio.
  promedioX = total / numLecturas;  //Se manda a la PC como un valor ASCII.
 //************************* Filtro eje Y *************************//
```

```
  .
  .
  .
 //*************************************************************************************//
  int X = promedioX;//Los datos son mapeados.
  X = map(X,-17,14,0,180);
  int Y = promedioY;
  Y = map(Y,-16,15,0,180);
  int Z = promedioZ;
  Z = map(Z,-15,17,0,180);
 //*****************************//
  asignaCodigo(X, Y, Z, Pila);  //Condiciones para la selección y envío de datos.
 }
}
```

3.3.1.1 Subrutinas

La primera subrutina invocada en el código principal del módulo esclavo es una llamada “asignaBuzzer”. En esta, se utiliza el valor del dato obtenido por la antena radiofrecuencia, y es aquí donde se realiza la configuración de los estados que debe tener el zumbador dispuesto en el módulo esclavo; es decir, si debe estar prendido o apagado. La figura 3.9 es de un diagrama de flujo en donde se muestra el funcionamiento de esta subrutina.

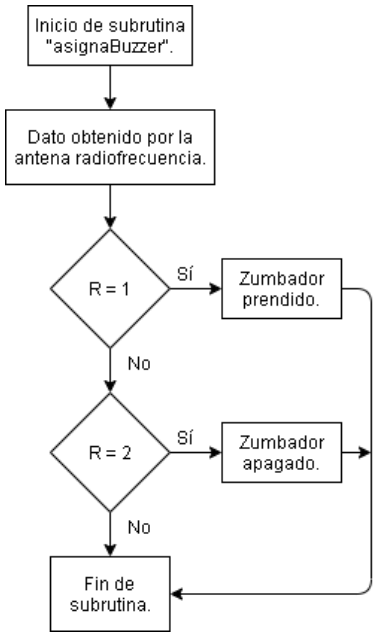

Figura 3.9 Diagrama de flujo de la subrutina "asignaBuzzer".

Esta subrutina, si bien no es compleja, ayuda a simplificar el programa principal. Aquí, se utiliza el valor del dato obtenido por la antena radiofrecuencia. Este valor es comparado con dos valores establecidos. Según coincida con alguno de los dos, se llevará a cabo la tarea asignada a ese valor establecido (prender/apagar el zumbador). Este es el código que lleva a cabo esta tarea:

```
void asignaBuzzer(int R)  //Subrutina de encendido-apagado del zumbador.
{
 if(R == 1)  //Condiciones para prender/apagar el buzzer.
 {
   Buz = true;
   return Buz;
 }
 else{
  if(R == 2)
  {
    Buz = false;
    return Buz;
  }
 }
}
```

En esta parte del programa, la subrutina inicia tomando el valor del dato obtenido por la antena radiofrecuencia, y lo almacena en la variable “R” como un *int*. Luego, compara este el valor del *int* con los valores ‘1’ y ‘2’, establecidos en el programa. Si el valor obtenido por la antena radiofrecuencia es igual a ‘1’, la variable “Buz”, declarada previamente como una variable global de tipo *bool*, que es con la que se define el estado del zumbador, tomará un valor de ‘true’, y regresará ese valor al programa principal. Si el valor obtenido por la antena radiofrecuencia es ‘2’, la variable “Buz” tomará el valor de ‘false’, y regresará ese valor al programa principal. Teniendo la variable “Buz” el valor de ‘true’, significa que el zumbador deberá estar encendido. La variable “Buz” con valor ‘false’ significa que el zumbador deberá estar apagado.

La siguiente subrutina invocada en el programa principal es la llamada “Bateria”. En esta subrutina es donde se hace la medición del nivel de voltaje que tiene la batería

que está alimentando al módulo esclavo. En la figura 3.10 se observa el diagrama de flujo que ejemplifica el funcionamiento de esta subrutina.

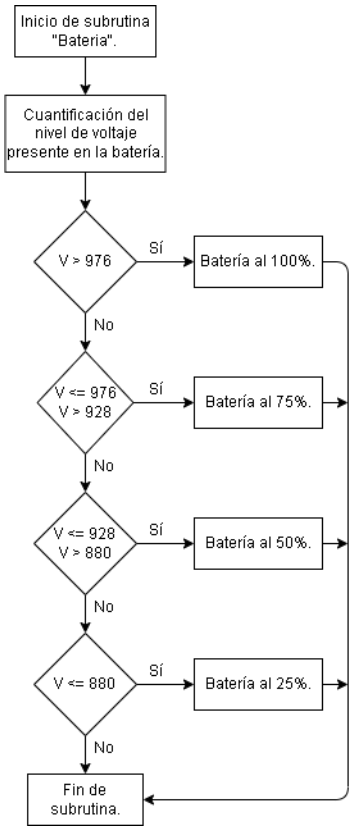

Figura 3.10 Diagrama de flujo para la subrutina "Bateria" del módulo esclavo.

Como ya se mencionó anteriormente, es en esta subrutina que se mide el nivel de voltaje con el que cuenta la batería que alimenta al módulo esclavo, y se hace una cuantificación de este voltaje, para luego ser comparado, en términos de bits, y así poder asignar un nivel de voltaje. Como no es posible obtener el voltaje de la batería desde su máxima capacidad, hasta estar prácticamente sin voltaje, se hace un escalamiento en el rango de voltaje con la que Arduino Nano y la antena radiofrecuencia puede operar sin problema. El código que lleva a cabo esta función es el siguiente:

```
void Bateria()
{
  float valor = analogRead(A0); //Convertidor analógico-digital.
  Valor = (valor * 1024)/3.7;
  //Condiciones de nivel de batería.
```

```
 if(Valor > 976.00)
 {
  Pila = 1;  //Asignación de código para nivel de batería.
 }
 else{
  if(Valor <= 976.00 && Valor > 928.00)
  {
   Pila = 2;    //Asignación de código para nivel de batería.
  }
  else{
   if(Valor <= 928.00 && Valor > 880.00)
   {
    Pila = 3;  //Asignación de código para el nivel de batería.
   }
   else{
    if(Valor <= 880)
    {
     Pila = 4;  //Asignación de código para el nivel de batería.
    }
   }
  }
 }
}
```

A diferencia de las otras subrutinas mencionadas hasta ahora, esta no inicia con algún dato tomado del programa principal. Lo primero es declarar la variable llamada "valor", de tipo *float*, a la que se le guardará los valores leídos por la entrada analógica/digital del Arduino Nano y, luego, estos valores serás convertidos a una escala de bits y almacenados en la variable "Valor", declarada como un variable global de tipo *float*. Después, se entra a una serie de condicionales *if*, en los que se compara el valor de bits obtenidos para la escala del nivel de voltaje en la batería contra valores establecidos dentro de los condicionales. Cabe mencionar que la variable "Pila" ha sido declarada también como una variable global, y el valor que es asignado a esta variable es utilizado después por una última subrutina.

La última subrutina es la llamada "asignaCodigo", y en esta es donde se determina la orientación del módulo esclavo, así como un código para el nivel de voltaje

presente en la batería que alimenta al módulo. La figura 3.11 muestra un diagrama de flujo del funcionamiento de esta subrutina, en la que se puede apreciar que cuenta con varias tomas decisiones.

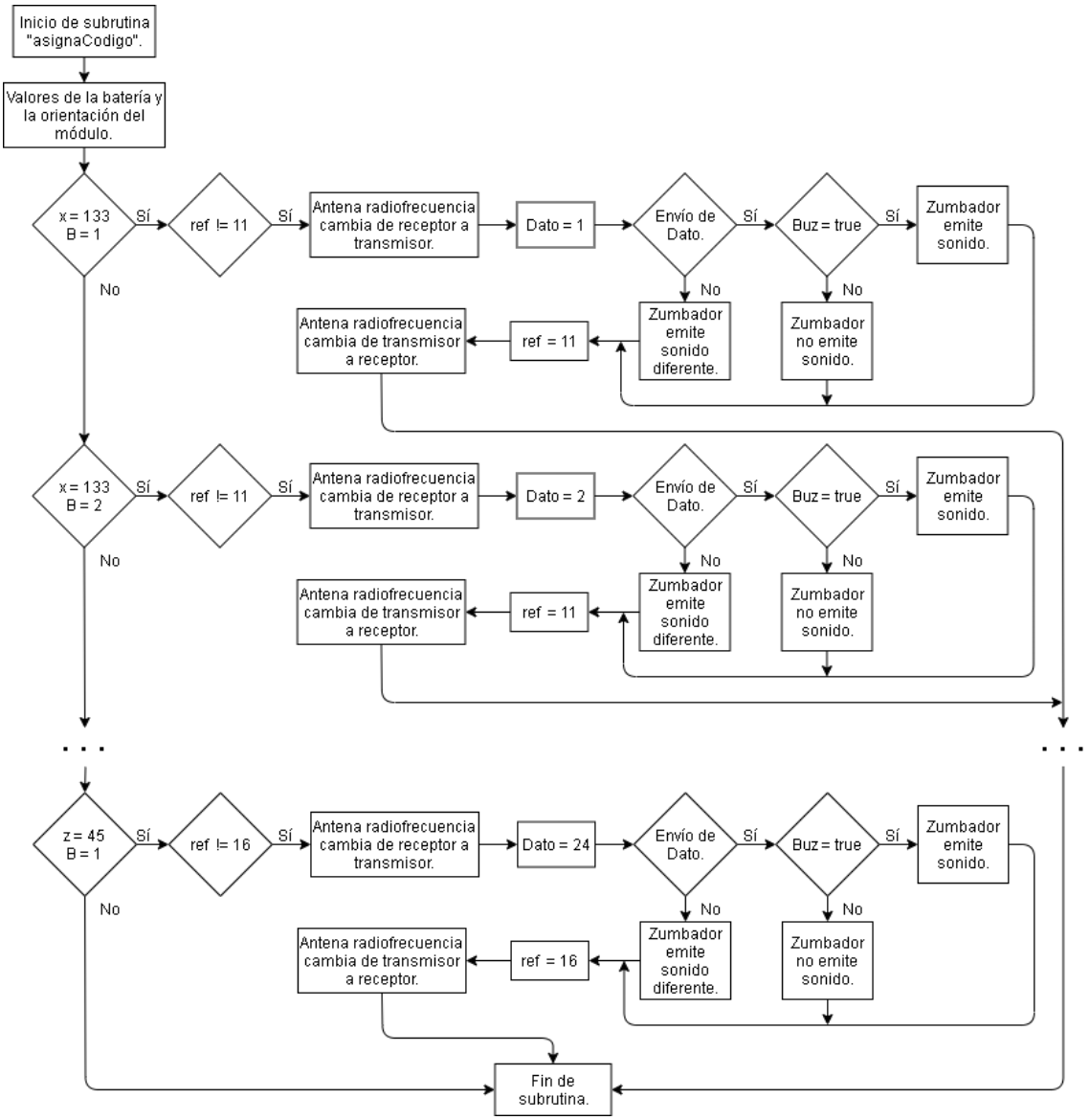

Figura 3.11 Diagrama de flujo para la subrutina "asignaCodigo".

Como se mencionó anteriormente en la sección del módulo maestro, los módulos esclavos cuentan con 24 posibles casos en los que se cumplen las condiciones necesarias para enviar al módulo maestro el nivel de voltaje en la batería y la orientación de los módulos esclavos. Estos 24 casos provienen del hecho de que, al comparar cada componente X, Y, X con respecto a los grados con los que se desea identificar el cambio de color en las tiras LED, también se hace la

comparación del valor que identifica el nivel de voltaje en la batería. Con esto, por cada componente X, Y, Z, se comparan cuatro valores para la batería. Al haber seis rangos a identificar para cada componente X, Y, Z, y por cada uno de esos seis están los cuatro valores a comparar para la batería, es que se obtienen los 24 posibles casos a cumplir. Aunque el prototipo desarrollado por (Mateos Peña, 2018) ya tenía identificados los rangos de inclinación para asignar el cambio de colores en las tiras LED, hubo que rectificar esos umbrales asignados para que los cambios de color funcionaran para este prototipo. El código que hace posible esta tarea es el siguiente:

```
void asignaCodigo(int x, int y, int z, int B)  //Condiciones para la selección y envío de datos.
{
 if(x == 133 && B == 1)  //1 X
 {
  if(ref!=11)
  {
   radio.stopListening();  //Se pasa al modo de transmisor.
   radio.openWritingPipe(PTXpipe); //Se abre el canal de escritura.
   DatoE = 1; //Se crea la variable con el dato del código correspondiente.
   //Serial.println(DatoE);
   if(radio.write(&DatoE, sizeof(DatoE)))  //Envío de datos.
   {
    if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
    else noTone(buz);            //valor retornado en la subrutina del mismo.
   }
   else{
    Serial.println("mal");  //Condición utilizada para pruebas. Sólo visible si se está conectado al PC.
   }
   ref = 11; //Se mantiene el estado de 'ref' hasta el siguiente cambio.
   delay(25);
   radio.startListening(); //Se regresa al modo de receptor.
  }
 }
 else
  if(x == 133 && B == 2)  //1 X
  {
```

```
 if(ref!=11)
 {
  radio.stopListening();  //Se pasa al modo de transmisor.
  radio.openWritingPipe(PTXpipe); //Se abre el canal de escritura.
  DatoE = 2; //Se crea la variable con el dato del código correspondiente.
  //Serial.println(DatoE);
  if(radio.write(&DatoE, sizeof(DatoE)))  //Envío de datos.
  {
   if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
   else noTone(buz);            //valor retornado en la subrutina del mismo.
  }
  else{
   Serial.println("mal");  //Condición utilizada para pruebas. Sólo visible si se está conectado al PC.
  }
  ref = 11; //Se mantiene el estado de 'ref' hasta el siguiente cambio.
  delay(25);
  radio.startListening(); //Se regresa al modo de receptor.
 }
}
.
.
.
    else
     if(z == 45 && B == 4) //6 z
     {
      if(ref!=16)
      {
       radio.stopListening(); //Se pasa al modo de transmisor.
       radio.openWritingPipe(PTXpipe);  //se abre el canal de escritura.
       DatoE = 24;  //Se crea la variable con el dato del código correspondiente.
       if(radio.write(&DatoE, sizeof(DatoE))) //Envío de datos.
       {
        if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
        else noTone(buz);            //valor retornado en la subrutina del mismo.
       }
       else{
        Serial.println("mal"); //Condición utilizada para pruebas. Sólo visible si se está conectado al PC.
```

```
            }
            ref = 16;  //Se mantiene el estado de 'ref' hasta el siguiente cambio.
            delay(25);
            radio.startListening();  //Se regresa al modo de receptor.
          }
        }
}
```

Como se puede observar, esta subrutina inicia con los valores obtenidos en el programa principal para los componentes X, Y, Z, así como el valor para el nivel de voltaje presente en la batería, obtenido de la subrutina “Bateria”. Estos valores son almacenados en las variables “x, y, z, B”, de tipo *int*, respectivamente. Después, viene la serie de estructuras de control *if* en las que se hace la comparación de los valores que tienen las variables utilizadas en esta subrutina contra el rango de valores establecidos para los componentes X, Y, Z, así como para el nivel de voltaje en la batería. En cuanto se cumple una estructura *if*, lo siguiente es otra estructura *if* dentro de la que fue cumplida, en la que se compara a la variable “ref”, declara como variable global de tipo *int*, utilizada como referencia del último caso con el que se realizó un envío de información al módulo maestro. Si se cumple esta sentencia, lo siguiente es que la antena radiofrecuencia cambie de rol, de receptor a transmisor, y se abre el canal de comunicación hacia el módulo maestro. Luego, a la variable llamada “DatoE” (declarada con anterioridad como variable global de tipo *int*) se le asigna un valor con el que se identifique las condiciones presentes en los módulos esclavos de la orientación y el nivel de batería en los mismos. Lo siguiente es que se ingresa a una nueva estructura *if* en la que se compara si el envío de la variable “DatoE” fue hecho o no. Si fue hecho, se compara el valor de la variable “Buz”, recordando que a esta variable se le asignó un valor en la subrutina “asignaBuzzer”, para saber si el zumbador debe o no emitir sonido cuando se haga el envío de esta variable hacia el módulo maestro. Si el envío de la variable “DatoE” no se hizo de forma correcta, el zumbador emite un sonido diferente, indicando el error en el envío de la información. Después, se cambia el valor de la variable “ref” para saber que ya se cumplió el caso que permitió el envío de información, y se vuelve a cambiar el rol de la antena radiofrecuencia, de transmisor a receptor.

3.3.2 Modelo esquemático y PCB

En la figura 3.12 se presenta el diagrama esquemático de las conexiones existentes entre los dispositivos que componen a los módulos esclavos, tales como el Arduino Nano, la antena radiofrecuencia, el acelerómetro, el módulo de carga, y el zumbador.

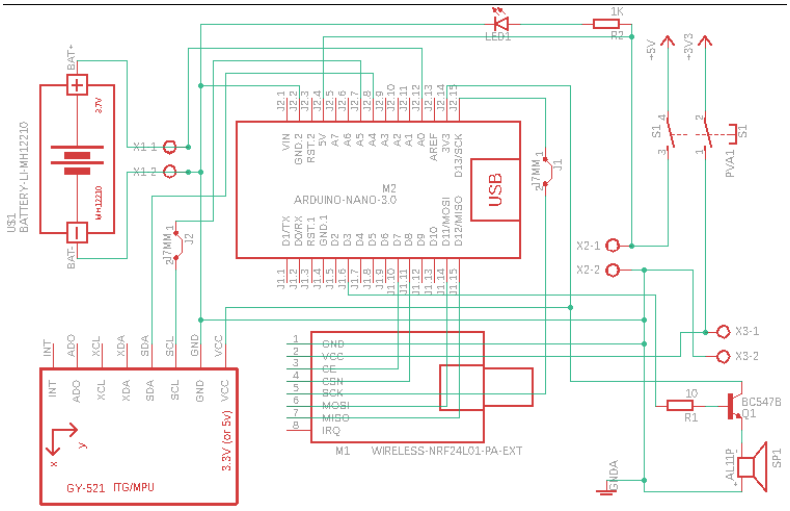

Figura 3.12 Diagrama esquemático de los módulos esclavos.

A partir de este diagrama esquemático, se obtuvo el modelo PCB para hacer la placa fenólica. Este modelo se muestra en la figura 3.13.

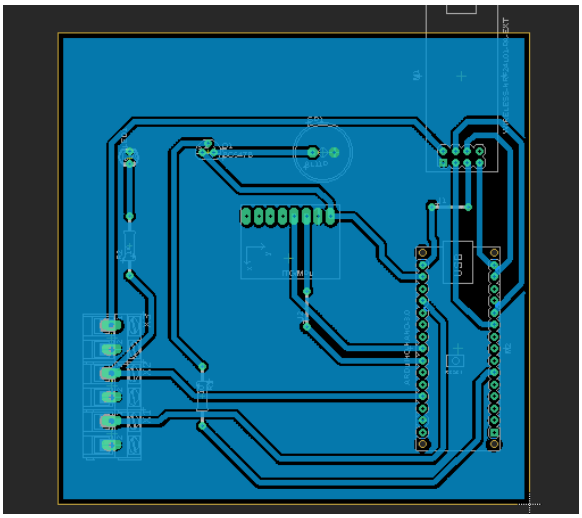

Figura 3.13 Diagrama PCB de los módulos esclavos.

3.4 Fuente de alimentación

Ya que el prototipo funciona en dos partes (módulo maestro y módulos esclavos), fueron necesarias diferentes fuentes de alimentación para cada módulo.

Para el maestro, se usó una fuente de 12 V y 240 W (figura 3.14). Fue necesario que contara con estas características, ya que la etapa de potencia para encender las tiras LED demanda 1.2 A por cada metro de tira LED, y 12 V. Al usar cuatro tiras LED de tres metros cada una, eso da un total de 172.8 W, por lo que la fuente de 240 W es más que suficiente para cubrir la demanda de potencia de las tiras LED.

Figura 3.14 Fuente de alimentación driver 12 V 20 A 240 W.

De esta misma fuente, se saca una derivación para alimentar al Arduino Mega presente en el módulo maestro, y otra más para alimentar a la antena radiofrecuencia. El hecho de que la antena radiofrecuencia no se conecte a la alimentación que proporciona Arduino Mega es debido a la corriente que demanda la antena radiofrecuencia para poder comunicarse con los módulos esclavos es muy alta con respecto a la que puede otorgar el Arduino Mega.

Para los módulos esclavos, se utilizó un cargador de batería *18650 Shield V3 Micro USB* (figura 3.15), en conjunto con un módulo de carga inalámbrico (figura 3.16). El cargador de batería es conectado al módulo de carga inalámbrico para poder reabastecer de voltaje a la batería que alimenta a los módulos esclavos sin tener que exponer el interior de los módulos esclavos.

Figura 3.15 Cargador de batería 18650 Shield V3.

Figura 3.16 Dispositivo receptor (izquierda) y dispositivo transmisor (derecha) para carga inalámbrica.

3.5 Desarrollo de la aplicación

La aplicación para dispositivos Android fue desarrollada en el entorno de *MIT App Inventor 2*. Para poder realizar la programación de la aplicación, primero fue necesario desarrollar un mapa de navegación, que sirvió de guía para obtener el prototipo final aquí mostrado. La figura 3.17 muestra este mapa de navegación, en el que se aprecian algunos componentes de la aplicación y su respuesta a la interacción con el usuario de la misma.

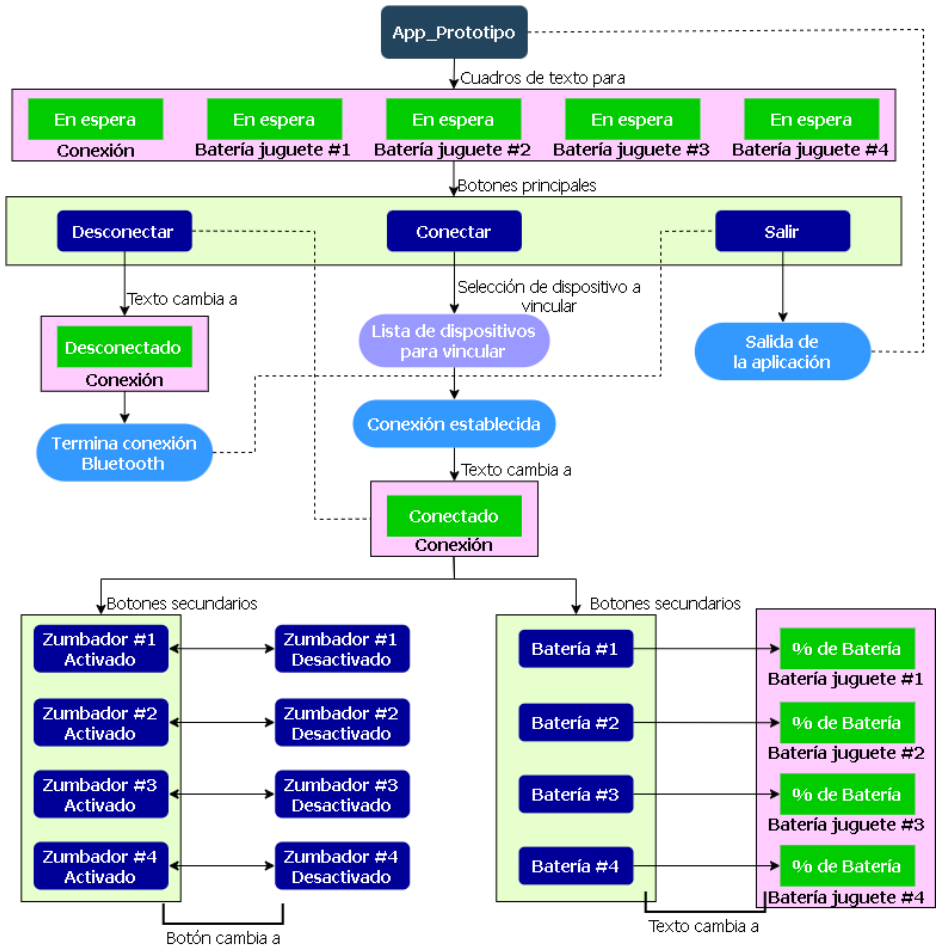

Figura 3.17 Mapa de navegación para la aplicación desarrollada para el prototipo.

Para iniciar la conectividad Bluetooth con el módulo maestro, se colocó un botón que permite escoger el dispositivo a vincular, así como uno más para desconectar la comunicación vía Bluetooth entre la aplicación y el módulo maestro. Con el propósito de manipular el encendido y apago de los zumbadores en los módulos esclavos, se utilizaron cuatro botones. Cada botón está identificado con un texto en donde se representa el número del módulo esclavo con el que se está interactuando, así como un cambio de color en el texto, dependiendo de si está encendido o apagado (de azul a rojo, y viceversa) el zumbador en los esclavos. También, se utilizaron cuatro botones más para mostrar el nivel de batería con el que cuentan los módulos esclavos, acompañados cada uno de un cuadro de texto, que es donde se mostrará el dato del nivel de batería en los módulos esclavos. Un botón más sirve para salir de la aplicación. Casi todos los botones están condicionados a sólo ejecutar la orden si es que hay conectividad vía Bluetooth con

el módulo maestro. La interfaz gráfica de la aplicación (figura 3.18) se encuentra inicializada con todos los zumbadores encendidos y sin mostrar el nivel de batería de los módulos esclavos, y siempre que sea abierta iniciará con estas características.

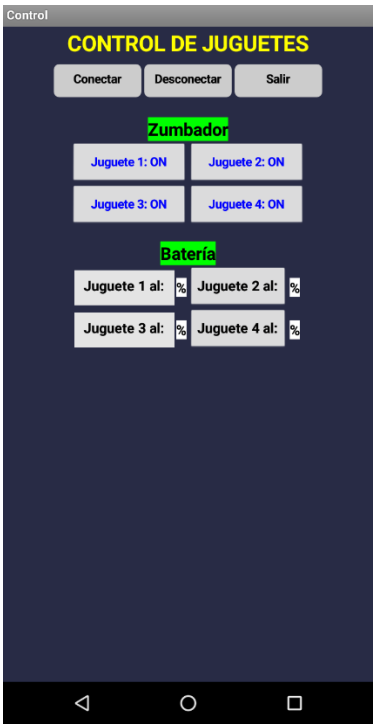

Figura 3.18 Vista de la aplicación para interactuar con los módulos esclavos.

Para que la aplicación pueda iniciar una conexión con la antena Bluetooth del módulo maestro se empleó un botón con el texto “Conectar” que, al ser presionado, muestra una lista con los nombres de los dispositivos que han sido vinculados previamente al dispositivo Android en el que está instalada la aplicación. De esta forma el usuario tendrá una lista de donde puede escoger el dispositivo a emparejar.

Lo que se muestra como un botón en la interfaz es en realidad una herramienta disponible en el entorno de *App Inventor 2* conocida como *ListPicker***,** a la cual se le debe especificar qué función realizar al ser presionada (*BeforePicking*) y después de ser presionada (*AfterPicking*). La figura 3.19 muestra la programación de la aplicación para la parte del inicio de la aplicación y la conexión Bluetooth.

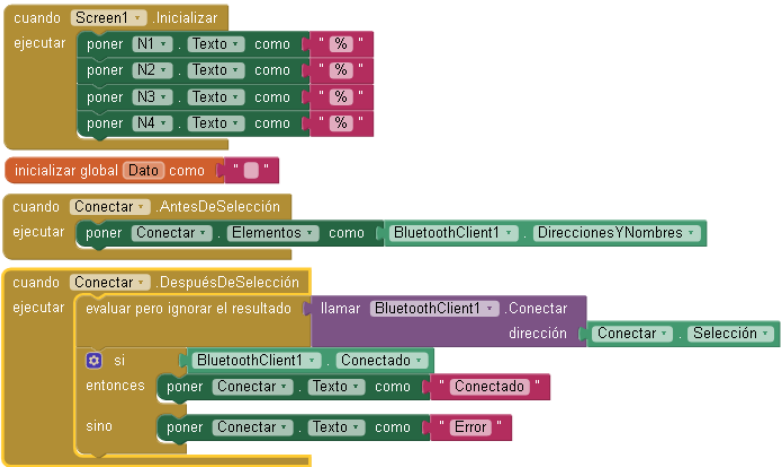

Figura 3.19 Parte del programa de la aplicación correspondiente al inicio de la misma y a la conectividad vía Bluetooth.

Cuando se presiona el botón “Conectar”, se evalúa si la conectividad Bluetooth del teléfono o tableta Android se encuentra activa. Si se da el caso, se mostrará la lista de todos los dispositivos anteriormente vinculados (figura 3.20). En caso contrario, la lista se mostrará vacía, lo que significa que no se encuentra activada la comunicación Bluetooth del dispositivo Android.

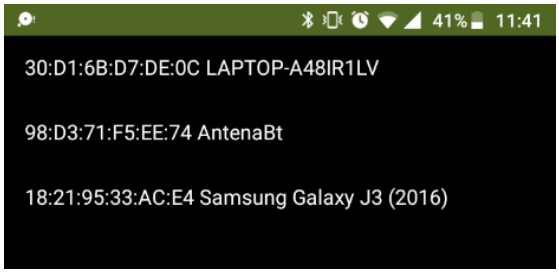

Figura 3.20 Interfaz de la aplicación mostrando los dispositivos previamente vinculados vía Bluetooth.

Después de presionar en el elemento de la lista al que se desea conectar (en este caso, la antena Bluetooth presente en el módulo maestro, identificada con el nombre de “AntenaBt”), se ordenará establecer la conexión con el mismo, y si el enlace fue exitoso, el texto en el botón de conectar cambia de “Conectar” a “Conectado” (figura 3.21). Esto se hace con el fin de informar al usuario, de forma visual, sobre el estado de la conexión entre la aplicación y el módulo maestro. Cuando se presiona el botón de “Desconectar”, el texto en el botón de conectar regresa a “Conectar”.

Figura 3.21 Vista de los cambios en el botón de conectar, según se establezca comunicación con la antena Bluetooth.

Los botones de encendido y apagado de los zumbadores en los módulos esclavos se mostrarán con el texto de color azul cuando el zumbador en el respectivo módulo esclavo se encuentre encendido, y con texto de color rojo cuando el zumbador se encuentre apagado (figura 3.22).

Figura 3.22 Cambio en los botones para los zumbadores, según sean encendidos o apagados.

Los botones para mostrar el nivel de batería en los módulos esclavos no cambian de color, puesto que vienen acompañados por un cuadro de texto. Este cuadro de texto muestra el nivel de batería presente en cada módulo esclavo (figura 3.23).

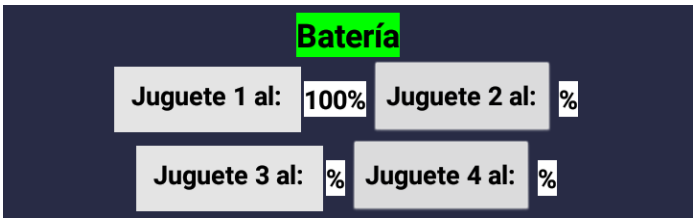

Figura 3.23 Cambios en las etiquetas de texto que muestran el nivel de batería en los módulos esclavos.

La programación para los botones de encendido y apagado de los zumbadores en los módulos esclavos, así como los botones con los que se pide mostrar el nivel de batería en cada módulo esclavo, se muestran en las figuras 3.24 y 3.25.

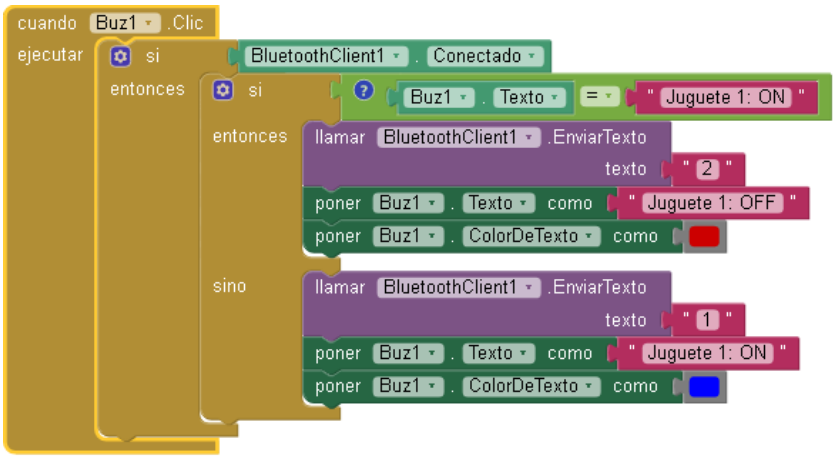

Figura 3.24 Parte del programa de la aplicación correspondiente al encendido y apagado de los zumbadores.

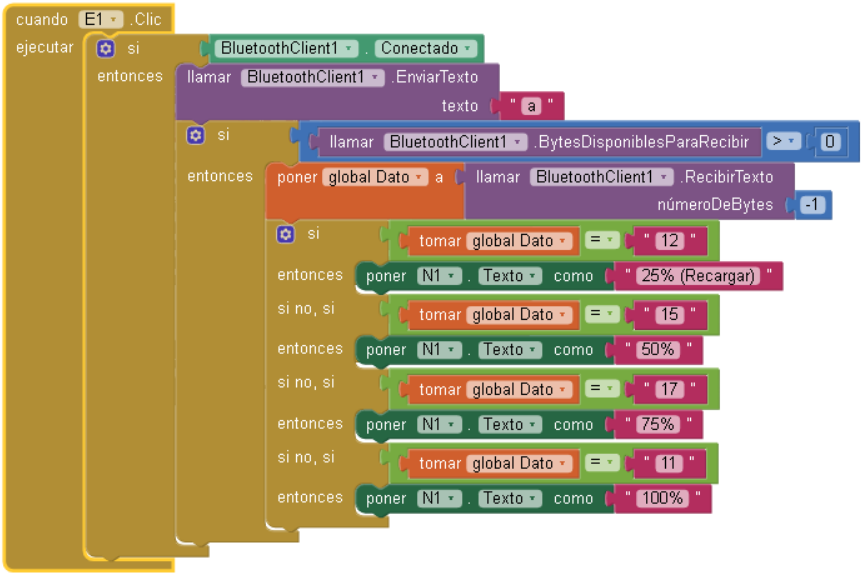

Figura 3.25 Parte del programa de la aplicación correspondiente a mostrar el nivel de batería en los módulos esclavos.

El botón de salida es, precisamente, para cerrar la aplicación. Un detalle es que, si se presiona el botón aun estando conectado al módulo maestro vía Bluetooth, la aplicación no se cerrará. Únicamente se podrá cerrar la aplicación al terminar la conexión Bluetooth con el módulo maestro. La figura 3.26 muestra la configuración hecha para este botón.

Figura 3.26 Configuración del botón para cerrar la aplicación.

3.6 Prototipo funcional

A continuación, se muestran algunas imágenes del prototipo terminado, identificando primero a los módulos esclavos (figuras 3.27 y 3.28), luego al maestro (figuras 3.29 y 3.30) y, por último, a todo el prototipo funcionando de forma uniforme (figura 3.31).

Figura 3.27 Módulo esclavo concluido.

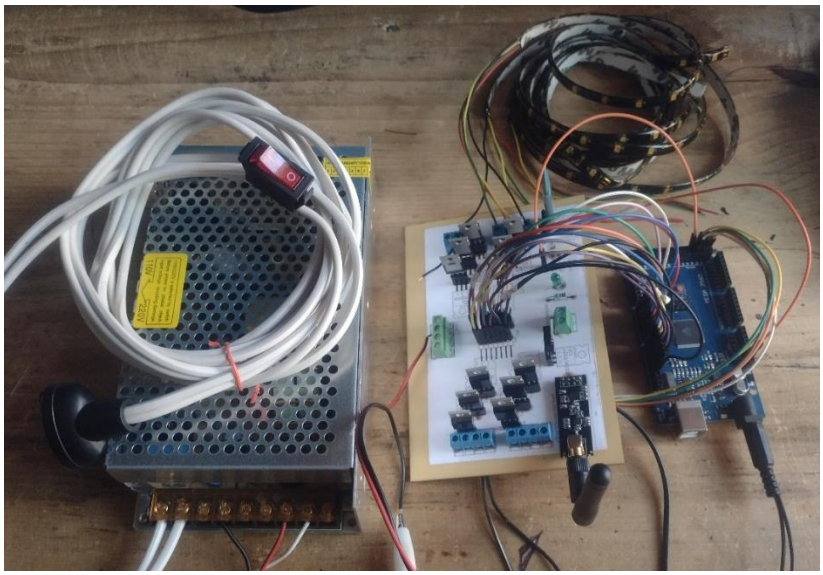

Figura 3.28 Módulo maestro concluido.

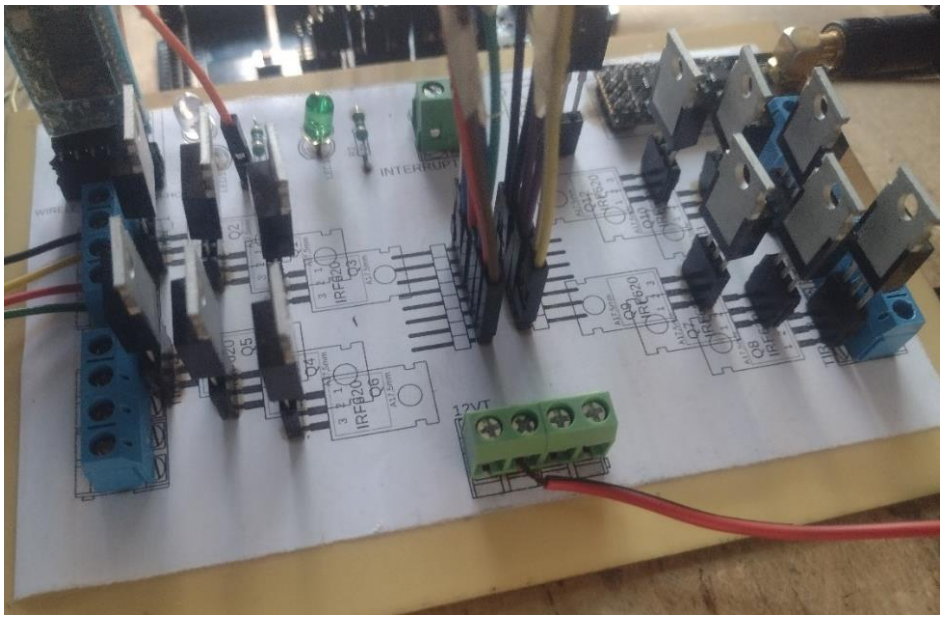

Figura 3.29 Vista de perfil de la placa del módulo maestro.

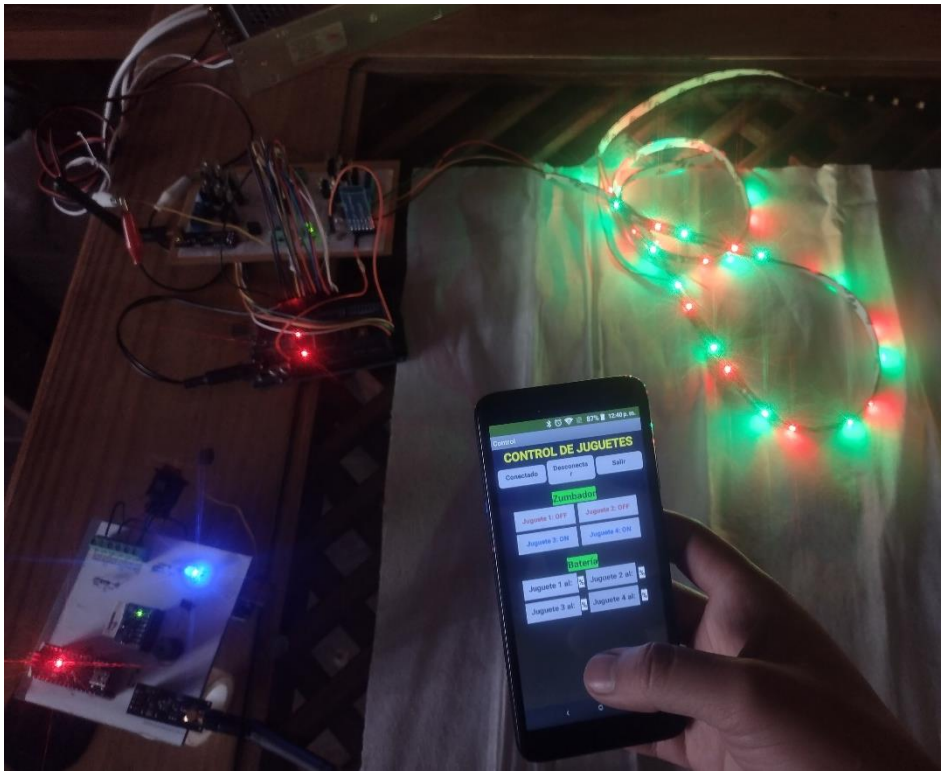

Figura 3.30 Módulos esclavo y maestro funcionando en conjunto con la aplicación y la etapa de potencia.

Capítulo 4 Pruebas y resultados

Ya que se partió del prototipo desarrollado por (Mateos Peña, 2018), ahora había que adecuar al sistema las nuevas propuestas mencionadas anteriormente. Para esto, primero fueron necesarias pruebas por separado para cada dispositivo, tanto para verificar su correcto funcionamiento, como para familiarizarse con su configuración.

Como la parte del intercambio bidireccional de información a través de radiofrecuencia es la principal forma de comunicación entre los módulos, se realizaron múltiples pruebas para conseguir todas las funciones requeridas, ya que los requerimientos en este sentido difieren respecto al modelo sobre el cual se trabajó, que solo utilizaba comunicación de esclavo a maestro. También, se hicieron pruebas para agregar la comunicación vía Bluetooth en conjunto con las pruebas para el desarrollo de la aplicación, con la que se puede configurar algunas funciones en los esclavos. También se llevaron a cabo pruebas para el acelerómetro.

Para la parte de la iluminación de las tiras LED en el módulo maestro, y la obtención del nivel de la batería que está alimentando al módulo esclavo y el zumbador presente en este último, sólo fueron necesarias pruebas, para después poder adicionar su función al sistema completo.

4.1 Comunicación por radiofrecuencia

Al proponer un sistema en el que es necesario el intercambio de información entre el maestro y los esclavos, se tuvo que adecuar un código de programación para dicha demanda. Para ello, lo primero fue comprobar que la comunicación bidireccional era posible.

Esto se comprobó utilizando dos códigos obtenidos de (Millervet, 2021). El primer código es para el módulo maestro, el cual, genera un número aleatorio (entero, del 0 al 10) y lo guarda en una variable, para después quedar en espera de que el módulo esclavo le envíe un número. Al recibir el dato, lo compara para ver si este coincide con el número aleatorio generado previamente, y si coincide, el maestro cambia de receptor a transmisor, y envía un mensaje de confirmación de

coincidencia al esclavo; de lo contrario, envía un mensaje de que el número es incorrecto.

El segundo código es para el módulo esclavo. En este, se genera un número aleatorio (entero, del 0 al 10) y se manda al módulo maestro. Después, el esclavo cambia de transmisor a receptor, e inicia un conteo de dos segundos. En este lapso de tiempo, si el esclavo recibe la confirmación de que el número que generó y mandó coincide con el que generó el maestro, finaliza su ejecución; de lo contrario, sigue enviando números hasta lograr generar y enviar uno que coincida con el del maestro. Ambos códigos están pensados para trabajar con un maestro y cuatro esclavos. Sin embargo, en esta prueba sólo se hizo uso de un maestro y un esclavo.

Para el módulo maestro si utilizó un Arduino Mega. El programa para el módulo maestro inicia con las bibliotecas “SPI, nRF24L01, RF24”, las cuales son necesarias para poder realizar la comunicación por radiofrecuencia. Después, se declara al nRF24 junto con los pines CE y CSN (pines digitales 7 y 8 del Arduino Mega, respectivamente), así como las direcciones de los nodos de comunicación. También, se declara la variable “daNumber” de tipo *byte*, en el que se guardará al número aleatorio generado. Esta es la codificación correspondiente de lo anteriormente descrito:

```
#include <SPI.h> //Call SPI library so you can communicate with the nRF24L01+
#include <nRF24L01.h> //nRF2401 libarary found at https://github.com/tmrh20/RF24/
#include <RF24.h> //nRF2401 libarary found at https://github.com/tmrh20/RF24/

const int pinCE = 7; //This pin is used to set the nRF24 to standby (0) or active mode (1)
const int pinCSN = 8; //This pin is used to tell the nRF24 whether the SPI communication is a command or message to send out
RF24 radio(pinCE, pinCSN); // Declare object from nRF24 library (Create your wireless SPI)

//Create up to 6 pipe addresses P0 - P5;  the "LL" is for LongLong type
const uint64_t rAddress[] = {0x7878787878LL, 0xB3B4B5B6F1LL, 0xB3B4B5B6CDLL, 0xB3B4B5B6A3LL, 0xB3B4B5B60FLL, 0xB3B4B5B605LL };

byte daNumber = 0; //The number that the transmitters are trying to guess
```

Luego, en la función *setup*, se hace uso de la función “randomSeed” para iniciar al generador de números pseudoaleatorios, y después se le asigna a la variable “daNumber” un número aleatorio entre el 0 y el 10. También, se inicia al monitor serial con un valor de 115200, y se hace mostrar un mensaje en donde se introduce al número con el que el esclavo debe coincidir. Lo siguiente es iniciar al nRF24, junto con algunas especificaciones en su funcionamiento, las cuales son que opere en un rango de bajo alcance, que trabaje en el canal 108, que inicie con la función de receptor, y declarando y abriendo los nodos de lectura de acuerdo al número de esclavos que van a estar funcionando. El código es el siguiente:

```
void setup()
{
  randomSeed(analogRead(0)); //create unique seed value for random number generation
  daNumber = (byte)random(11); //Create random number that transmitters have to guess
  Serial.begin(115200);  //start serial to communication
  Serial.print("The number they are trying to guess is: ");
  Serial.println(daNumber); //print the number that they have to guess
  Serial.println();
  radio.begin();  //Start the nRF24 module
  radio.setPALevel(RF24_PA_LOW);  // "short range setting" - increase if you want more range AND have a
good power supply
  radio.setChannel(108);          // the higher channels tend to be more "open"

  // Open up to six pipes for PRX to receive data
  radio.openReadingPipe(0,rAddress[0]);
    .
    .
    .
  radio.openReadingPipe(5,rAddress[5]);
  radio.startListening();                 // Start listening for messages
}
```

Siguiendo con la función *loop*, lo primero es declarar las variables “pipeNum, gotByte”, de tipo *byte*; la primera servirá para identificar al esclavo que hizo el envío del número aleatorio (valores del 1 al 4), y la segunda servirá para almacenar dicho número. Después, se entra a una estructura de control *while*, en la que se especifica

que, mientras exista información recibida, se deberán realizar todas las líneas de código que se encuentren dentro de ese *while*. En estas líneas de código, lo primero es leer y guardar el número aleatorio recibido. Luego, en un mensaje se muestra el número recibido y el esclavo que lo mandó, para después hacer la comparación entre el número generado por el maestro y el número recibido de parte del esclavo. Si los números no coinciden uno con el otro (no son iguales), se hace saber a través de un mensaje en el monitor serial. Pero en el caso de que los números coincidan entre sí (ambos son iguales), se hace otra comparación en la que se entra a una subrutina, llamada “sendCorrectNumber”, a la que se le envía el valor del esclavo que envió el número que coincidió con el que generó el maestro. Una vez termina de ejecutarse la subrutina, el resultado de la comparación imprime un mensaje que confirma que el esclavo generó un número igual al que generó el maestro, o avisa que el envío de ese mensaje de confirmación no pudo ser hecho. Este fue el código utilizado:

```
void loop()
{
   byte pipeNum = 0; //variable to hold which reading pipe sent data
   byte gotByte = 0; //used to store payload from transmit module

   while(radio.available(&pipeNum)){ //Check if received data
    radio.read( &gotByte, 1 ); //read one byte of data and store it in gotByte variable
    Serial.print("Received guess from transmitter: ");
    Serial.println(pipeNum + 1); //print which pipe or transmitter this is from
    Serial.print("They guess number: ");
    Serial.println(gotByte); //print payload or the number the transmitter guessed
    if(gotByte != daNumber) { //if true they guessed wrong
    Serial.println("Fail!! Try again.");
    }
    else { //if this is true they guessed right
     if(sendCorrectNumber(pipeNum)) Serial.println("Correct! You're done."); //if true we successfully
responded
     else Serial.println("Write failed"); //if true we failed responding
    }
    Serial.println();
```

```
    }
  delay(200);
}
```

En la subrutina llamada “sendCorrectNumber” es donde se lleva a cabo el cambio de rol del módulo maestro, pasando de receptor a transmisor. Una vez dentro de la subrutina, se inicia una variable de tipo *bool* llamada “worked”, que servirá para indicar al condicional en donde se encuentra esta subrutina, en el programa principal, si el envío del mensaje de confirmación fue exitoso o no. Después, se cambia a modo de transmisor deteniendo el modo de receptor, y se abre el nodo de escritura. Este nodo será el mismo por el que llegó el número correcto, recordando que el nodo se obtuvo en el programa principal y fue dirigido a esta subrutina. Luego, viene un condicional, en el que se hace enviar al esclavo el número que generó el maestro, además de asignar un valor a la variable “worked”; si el envío fue realizado, la variable toma el valor de ‘true’, y de no ser realizado, la variable toma el valor de ‘false’. Por último, se regresa al modo de receptor deteniendo el modo de transmisor, y se retorna a la variable “worked” al programa principal, justo donde se hizo el salto a la subrutina. El código para esta subrutina es el siguiente:

```
//This function turns the receiver into a transmitter briefly to tell one of the nRF24s
//in the network that it guessed the right number. Returns true if write to module was
//successful
bool sendCorrectNumber(byte xMitter) {
    bool worked; //variable to track if write was successful
    radio.stopListening(); //Stop listening, start receiving data.
    radio.openWritingPipe(rAddress[xMitter]); //Open writing pipe to the nRF24 that guessed the right number
      // note that this is the same pipe address that was just used for receiving
    if(!radio.write(&daNumber, 1))  worked = false; //write the correct number to the nRF24 module, and
check that it was received
    else worked = true; //it was received
    radio.startListening(); //Switch back to a receiver
    return worked;  //return whether write was successful
}
```

Para el módulo esclavo se utilizó un Arduino Nano. El programa para el módulo esclavo inicia con las librearías “SPI”, “nRF24L01” y “RF24”, que son necesarias para poder llevar a cabo la comunicación por radiofrecuencia. También, se declara al nRF24, sus pines CE y CSN (pines digitales 7 y 8 del Arduino Nano, respectivamente), su nodo de comunicación, y la dirección de dicho nodo. También, se genera una variable de tipo *byte* llamada “counter” con valor de 1, y otra de tipo *bool* llamada “done” con valor de ‘falso’. Este es el código:

```
#include <SPI.h> //Call SPI library so you can communicate with the nRF24L01+
#include <nRF24L01.h> //nRF2401 libarary found at https://github.com/tmrh20/RF24/
#include <RF24.h> //nRF2401 libarary found at https://github.com/tmrh20/RF24/


const int pinCE = 7; //This pin is used to set the nRF24 to standby (0) or active mode (1)
const int pinCSN = 8; //This pin is used to tell the nRF24 whether the SPI communication is a command or message to send out
RF24 radio(pinCE, pinCSN); // Create your nRF24 object or wireless SPI connection
#define WHICH_NODE 2     // must be a number from 1 - 6 identifying the PTX node
const uint64_t wAddress[] = {0x7878787878LL, . . . ,0xB3B4B5B605LL};
const uint64_t PTXpipe = wAddress[ WHICH_NODE - 1 ];   // Pulls the address from the above array for this node's pipe
byte counter = 1; //used to count the packets sent
bool done = false; //used to know when to stop sending packets
```

En la función *setup*, se inicia al puerto serial con un valor de 115200, y también se hace uso de la función “randomSeed” para iniciar al generador de números pseudoaleatorios. Luego, se inicia al nRF24, así como se definen algunas de sus funciones, como lo son que trabaje en un rango de bajo alcance, que lo haga a través del canal 108, declarando y abriendo el nodo de recepción, e iniciando con la función de receptor:

```
void setup()
{
  Serial.begin(115200);   //start serial to communicate process
  randomSeed(analogRead(0)); //create unique seed value for random number generation
  radio.begin();           //Start the nRF24 module
  radio.setPALevel(RF24_PA_LOW);  // "short range setting" - increase if you want more range AND have a good power supply
```

```
  radio.setChannel(108);          // the higher channels tend to be more "open"
  radio.openReadingPipe(0,PTXpipe);  //open reading or receive pipe
  radio.stopListening(); //go into transmit mode
}
```

Luego, en la función *loop*, se empieza con un condicional en el que compara el valor de la variable “done”; cuando esta variable tiene valor ‘false’, se lleva a cabo todo el código que se encuentre dentro del condicional, pero cuando tiene por valor ‘true’ no lleva a cabo ninguna otra acción. Dentro del condicional se declara la variable “randNumber” de tipo *byte*, y en esta variable se asigna el número aleatorio. Luego, se abre el nodo de escritura, y se envía el número aleatorio a través de un segundo condicional. Si no se envía el número, se informa en el monitor serial con un mensaje de error de envío, pero si sí se envía el número, se informa con un mensaje de confirmación en el monitor serial, y se hace cambiar de rol, de transmisor a receptor, al módulo esclavo. Después, se inicia una ventana de tiempo de 200 ms, en la que se debe recibir la confirmación por parte del módulo maestro de que el número que el esclavo generó y mandó coincide con el número que el maestro generó. Esto último se logra declarando dos variables, una de tipo *unsigned long* y otra de tipo *bool*. La primera se declara con el nombre de “startTimer” y se le asigna el valor que retorne la función “millis()” (esta función retorna el tiempo, en milisegundos, que lleva prendido el Arduino), y la segunda se declara con el nombre de “timeout” y se le asigna ‘false’ como valor. Una vez se tienen estas variables, se entra a una estructura *while* en el que se mantiene dentro mientras no exista recepción de información en el nRF24 y la variable “timeout” mantenga el valor de ‘false’. Dentro de esta *while*, se encuentra un condicional en el que se verifica que el resultado de restar, al dato retornado por la función “millis()” en ese condicional, el valor de tiempo obtenido previamente en la variable “startTimer”, sea mayor a 200 ms. Si el resultado de esta resta es mayor a 200 ms, a la variable “timeout” se le asigna ‘true’ como valor. Con esto, cuando la variable “timeout” tiene ‘true’ como valor, se sale de esa *while*, pero hay que tener en cuenta de que el recibir el mensaje de confirmación de parte del módulo maestro antes de que pase el lapso de tiempo

de 200 ms también hará salirse de esa *while*, con el valor de la variable “timeout” como ‘false’. Este es el código:

```
void loop()
{
  if(!done) { //true once you guess the right number
    byte randNumber = (byte)random(11); //generate random guess between 0 and 10
    radio.openWritingPipe(PTXpipe);        //open writing or transmit pipe

    if (!radio.write( &randNumber, 1 )){  //if the write fails let the user know over serial monitor
       Serial.println("Guess delivery failed");
    }
    else { //if the write was successful
        Serial.print("Success sending guess: ");
        Serial.println(randNumber);

      radio.startListening(); //switch to receive mode to see if the guess was right
      unsigned long startTimer = millis(); //start timer, we will wait 200ms
      bool timeout = false;
      while ( !radio.available() && !timeout ) { //run while no receive data and not timed out
        if (millis() - startTimer > 200 ) timeout = true; //timed out
      }
      if (timeout) Serial.println("Last guess was wrong, try again"); //no data to receive guess must have been
wrong
      else  { //we received something so guess must have been right
        byte daNumber; //variable to store received value
        radio.read( &daNumber,1); //read value
        if(daNumber == randNumber) { //make sure it equals value we just sent, if so we are done
          Serial.println("You guessed right so you are done");
          done = true; //signal to loop that we are done guessing
        }
        else Serial.println("Something went wrong, keep guessing"); //this should never be true, but just in case
      }
      radio.stopListening(); //go back to transmit mode
    }
  }
   delay(1000);
}
```

Como se puede observar, una vez fuera de la estructura *while* lo siguiente es un condicional en el que se compara el valor de la variable “timeout”. En caso de tener como valor ‘true’, muestra en el monitor serial un mensaje de que no hubo respuesta por parte del módulo maestro, y que volverá a mandar un número aleatorio. Pero si el valor de la variable “timeout” es ‘false’, se declara una variable llamada “daNumber” de tipo *byte*, en la que se guardará el número que mandó el módulo maestro. Entonces, se entra a otro condicional, en el que se compara al número recibido por parte del módulo maestro con el número aleatorio que el módulo esclavo generó previamente. Si no coinciden los números entre sí (no son iguales), se hace ver un mensaje en el monitor serial de que los números no son iguales. En caso de coincidir ambos números (son iguales), se muestra en el monitor serial un mensaje que confirma que ambos números son iguales, y a la variable “done” se le asigna ‘true’ como valor. Una vez realizada esta comparación, el esclavo retoma su rol de transmisor. Por último, recordemos que el primer condicional dentro de la función *loop* comparaba el valor de la variable “done”, en donde, si esta variable tuviera como valor ‘false’ llevaría a cabo el resto de código que se encontrara dentro del condicional. Pero una vez que la variable “done” tiene el valor ‘true’, se pasa al final del condicional, que es también el final del código principal, por lo que el esclavo detiene su función de generar números aleatorios y enviarlos al maestro. Los circuitos usados para estas pruebas se observan en la figura 4.1.

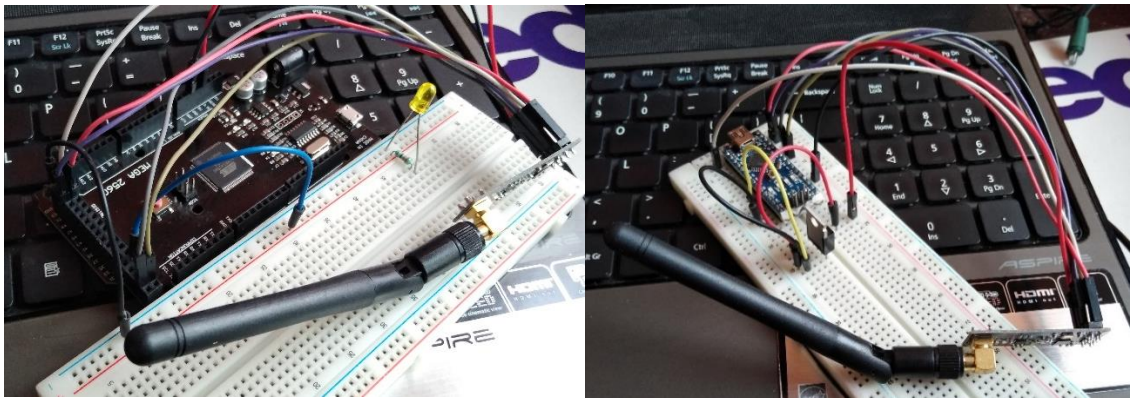

Figura 4.1 Circuitos para el módulo maestro (izquierda) y el módulo esclavo (derecha), utilizados en las pruebas de comunicación bidireccional por radiofrecuencia.

Al servir estos códigos como experimento, lo siguiente fue corroborar si el intercambio de información se veía interrumpido por alguna tarea más compleja, y también se comprobó qué tipo de datos se podían usar en la comunicación.

Para esto, se generaron dos códigos que parten de los previamente mostrados, y que uno sirve para un módulo maestro y otro para un módulo esclavo, y se usaron los mismos circuitos que habían sido usados para las pruebas de comunicación bidireccional.

Para el módulo maestro, el código permite verificar si ha recibido información. Cabe mencionar que ahora se trabajó con cadenas de datos, por lo que, una vez almacenada la información recibida, se pasa a separarla, de forma que se puedan realizar tareas subsecuentes con los respectivos datos separados. En este caso, sólo se ocupó el primer dato separado para realizar una subrutina, mientras que el segundo dato simplemente se muestra en el monitor serial. Esto para comprobar que la recepción y separación de datos es correcta. Este código inicia con las bibliotecas “SPI”, “nRF24L01” y “RF24”, como se muestra a continuación:

```
#include <SPI.h> //Biblioteca para comunicarse con el nRF24.
#include <nRF24L01.h>
#include <RF24.h> //Bibliotecas para el nRF24.
```

Después, en la función *setup*, se inicia al puerto serial, se hacen las configuraciones para que la antena radiofrecuencia opere de forma correcta, y se declara una salida para un LED que servirá más adelante como medio visual para comprobar que los módulos maestro y esclavo interactúan entre sí:

```
void setup()
{
  Serial.begin(9600);  //start serial to communication
  radio.begin();  //Start the nRF24 module
  radio.setPALevel(RF24_PA_MIN);  // "short range setting" - increase if you want more range AND have a good power supply
  radio.setChannel(108);          // the higher channels tend to be more "open"
  // Open up to six pipes for PRX to receive data
  radio.openReadingPipe(0,dRadio[0]);
  radio.openReadingPipe(1,dRadio[1]);
  radio.openReadingPipe(2,dRadio[2]);
```

```
 radio.openReadingPipe(3,dRadio[3]);
 radio.startListening();          // Start listening for messages
 pinMode(led, OUTPUT);
}
```

En la función *loop*, lo primero es comprobar que el módulo maestro ha recibido información por parte del esclavo. En caso de que haya información recibida, se genera una variable llamada “pipeNum”, de tipo *byte*, y con valor de ‘0’. Luego, se entra a una estructura *while*, en la que se hace la recepción de la cadena de datos enviada por el esclavo. Dentro de esta función es donde se separa la cadena de datos en dos valores, y estos valores se asignan a una variable diferente cada uno. Estas dos variables son de tipo *char*, y tiene un tamaño de 3 caracteres. Una vez cargados estos valores a las variables, se cambia el tipo de variable; es decir, se pasa de ser *char* a ser *int*, con el fin de comprobar si una conversión de este tipo es posible, y si se pueden usar después estos valores convertidos. Una vez convertidas las variables, se imprimen en el monitor serial, y luego se hace una invocación a dos subrutinas. El código que corresponde a esta sección es el siguiente:

```
void loop()
{
 if(radio.available())
 {
   byte pipeNum = 0; //Variable para saber cuál esclavo mandó información.
   while(radio.available(&pipeNum))
   { //Verificación de recepción de datos.
     int len = 0;
     char DatoR[5];
     len = radio.getDynamicPayloadSize();
     radio.read(&DatoR, len ); //Lectura y almacenamiento de datos.
     Serial.print("Recibido de: ");
     Serial.println(pipeNum + 1); //Muestra al esclavo que envío la información.
     Serial.print("Muestra de: "); //Impresión de información recibida.
     Serial.println(DatoR);

     const char Clave[] = ","; //Caracter clave para separación de trama de datos.
     char *Llave = strtok(DatoR, Clave); //Identificación de caracter clave en la trama de datos.
```

```
    strncpy(Valor1, Llave, sizeof(Valor1));
    Llave = strtok(NULL, Clave);
    strncpy(Valor2, Llave, sizeof(Valor2)); //Separación de datos.

    int convertido = String(Valor1).toInt();  //Conversión de variables para un manejo más sencillo.
    int Convertido = String(Valor2).toInt();
    Serial.println(convertido);
    Serial.println(Convertido);

    prendeLed(convertido);  //Subrutina de prueba.

    if(prendido)
    {
      if(Enviado(pipeNum))  Serial.println("Hecho");
      else  Serial.println("Incompleto");
    }
   }
  }
  delay(200);
}
```

En la primera subrutina se hace prender un LED durante un lapso, según el dato obtenido en la segmentación de la cadena de datos. Una vez prendido este LED, se retorna al programa principal un dato de tipo *booleano* iniciado como ‘false’, y que cambia a ‘true’ una vez el LED se prende y se apaga. Este dato booleano servirá para avisar al programa principal que la subrutina se realizó de forma correcta. De vuelta en el programa principal, se encuentra un condicional que utiliza el dato booleano retornado por la subrutina. Si el booleano retorna como ‘true’, se entra a un segundo condicional, en el que también salta a una segunda subrutina. El código de la primera subrutina es el siguiente:

```
void prendeLed(int Valor1)
{
  if(Valor1 == "3")
  {
    digitalWrite(led, HIGH);
    delay(3000);
```

```
   digitalWrite(led, LOW);
  }
  else{
   if(Valor1 == '3')
   {
    digitalWrite(led, HIGH);
    delay(2000);
    digitalWrite(led, LOW);
   }
   else{
    if(Valor1 == 11)
    {
     digitalWrite(led, HIGH);
     delay(1000);
     digitalWrite(led, LOW);
     prendido = !prendido;
     return prendido;
    }
   }
  }
}
```

La segunda subrutina es la que hace cambiar al maestro de receptor a transmisor. Primero se genera otra variable de tipo *bool*, cuyo valor será checado por el condicional que hizo saltar a esta subrutina. Lo siguiente que hace es que envía el valor de la variable “E” para hacer saber al esclavo que se ha llevado la tarea de forma correcta. Según se haya enviado o no dicho dato, la variable booleana toma un valor y lo retorna al programa principal, el cual nos indicará en el monitor serial si el envío fue exitoso o no. Este es el código para esta subrutina:

```
bool Enviado(byte xMitter) {
   bool done; //Variable para confirmar el envío de información.
   radio.stopListening(); //Maestro pasa a ser trasmisor.
   radio.openWritingPipe(dRadio[xMitter]); //Abre el canal de escritura hacia el esclavo que activó la tarea correcta.
   if(!radio.write(&E, 1))  done = false; //Escribe el dato a enviar, y verifica que haya sido envíado.
   else done = true; //Recepción exitosa por parte del esclavo.
   radio.startListening(); //Maestro regresa a ser receptor.
```

```
   return done;  //Retorna el valor a la rutina principal.
}
```

Para el esclavo, el código concatena dos variables de tipo *char* que contienen valores, que servirán como códigos para realizar subrutinas en el maestro. El primer valor incluye una coma que servirá para que el maestro separe la cadena de datos. Una vez concatenados los datos, los envía al maestro, y muestra en el monitor serial un mensaje de confirmación para un envío exitoso, o un envío fallido, en caso de no haber podido enviar la cadena de datos. El código inicia con las bibliotecas “SPI”, “nRF24L01”, “RF24” y “String”, como se muestra a continuación:

```
#include <SPI.h> //Biblioteca para comunicarse con el RF24.
#include <nRF24L01.h>
#include <RF24.h> //Bibliotecas para el RF24.
#include <String.h> //Biblioteca para poder trabajar con cadenas de datos.
```

En la parte del *set up*, se inicia al puerto serial y se configura a la antena radiofrecuencia:

```
void setup()
{
  Serial.begin(9600);
  radio.begin();  //Inicia el nRF.
  radio.setPALevel(RF24_PA_MIN);
  radio.setChannel(108);
  radio.openReadingPipe(0,PTXpipe);  //Abriendo canal de lectura.
  radio.stopListening(); //Módulo inicia como transmisor.
}
```

A diferencia del maestro, para el código del esclavo no se recurrió al uso de las subrutinas. Simplemente, si se cumple la condición en donde se confirma el envío de información, el esclavo cambia de transmisor a receptor. Al cambiar de rol, se hace un proceso en el que el esclavo inicia un conteo de dos segundos. Durante este lapso de tiempo, el esclavo se mantiene a la espera de que el maestro envíe un mensaje de confirmación de que realizó el encendido del LED. Si pasan los dos segundos y el esclavo no recibe respuesta de parte del maestro, muestra en el monitor serial un mensaje de una posible falla en el maestro. Cuando sí recibe una respuesta, se muestra un mensaje de que se cumplió con el ciclo del sistema.

Cuando es mostrado el mensaje de confirmación, el esclavo deja de enviar datos al maestro. Este es el código utilizado:

```
void loop()
{
 if(!done)
 { //Verdadero una vez que envía información.
   char Dato1[] = "11,";
   char Dato2[] = "12"; //Datos a enviar.
   strcat(Dato1, Dato2); //Formando trama de datos.
   radio.openWritingPipe(PTXpipe); //Abriendo canal de escritura.

   if (!radio.write(&Dato1, strlen(Dato1)))
   {  //Si el envío de información falla, lo informa en el monitor Serial.
    Serial.println("Fallido");
   }
   else
   { //Si el envío se realiza de forma correcta.
    Serial.print("Se envió: ");
    Serial.println(Dato1);

    radio.startListening(); //Cambia al modo de receptor.
    unsigned long startTimer = millis(); //Inicia conteo de tiempo para poder esperar 2 seg.
    bool timeout = false;
    while ( !radio.available() && !timeout )
    { //Mientras no reciba datos y no hayan pasado los 2 seg..
     if (millis() - startTimer > 2000 ) timeout = true; //Pasados los 2 seg.
    }
    if (timeout) Serial.println("Posible falla en maestro"); //No hay datos recibidos, posible falla del maestro.
    else
    { //Recepción de información correcta.
     byte daNumber; //Variable para almacenar el dato recibido.
     radio.read( &daNumber, sizeof(daNumber));

     if(daNumber == Hecho)
     {
      Serial.println("Hecho");
```

```
        done = true; //Concluimos el envío de datos.
       }
       else Serial.println("Fallido");
      }
     radio.stopListening(); //Se regresa al modo de transmisión.
    }
  }
 delay(2000);
}
```

4.2 Orientación de los módulos esclavos

Con la intención de comprobar si era posible optimizar el proceso de la obtención de la orientación de los esclavos o trabajar con acelerómetros de menor costo, se optó por utilizar otro dispositivo acelerómetro.

Utilizando el MPU6050, se llevaron a cabo pruebas de su funcionamiento y del cómo trabajar con la información que proporcionaba. Con esto se obtuvo el circuito que se muestra en la figura 4.2, con el que se pudieron realizar las primeras pruebas.

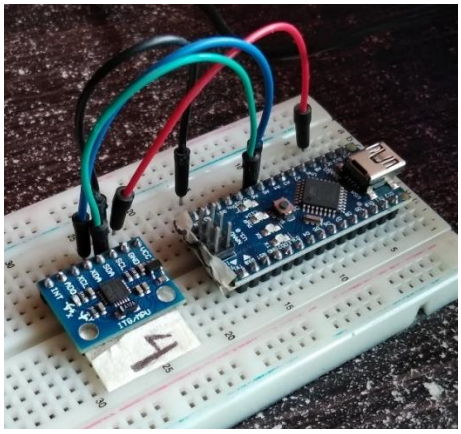

Figura 4.2 Circuito utilizado para las pruebas con el MPU6050.

Lo primero fue calibrar los acelerómetros, ya que existe una posibilidad de que, al ser soldar el sensor a la placa fenólica, pueda quedar desnivelado (defecto de fábrica), provocando un error en cada componente X, Y, Z, así como también se pudiera colocar de forma desnivelada en el prototipo final. Para esto, se utilizó el código obtenido de (Naylamp Mechatronics, 2020) en conjunto con una aplicación para teléfono celular (figura 4.3), cuya función es la de mostrar los grados de

inclinación utilizando el giroscopio con el que cuenta el teléfono. En la figura 4.4 se muestra cómo se calibró al MPU6050 con estas dos herramientas.

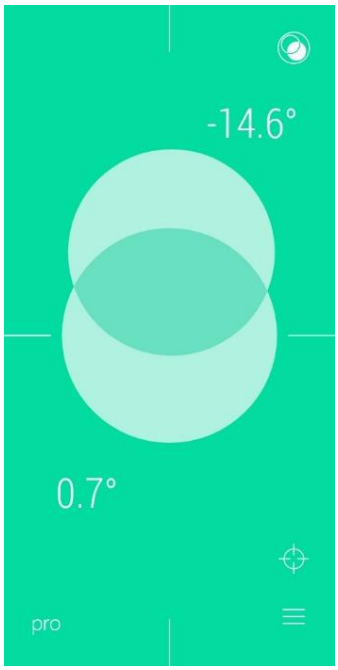

Figura 4.3 Vista de la aplicación "Nivel de burbuja" , utilizada para auxiliar la calibración del MPU6050.

Figura 4.4 Calibración del MPU6050.

El código inicia con las bibliotecas para poder controlar al MPU-6050, que son “I2Cdev”, “MPU6050” y “Wire”, además de algunas variables necesarias para almacenar los valores “x, y, z,”, como se muestra a continuación:

```
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"
// La dirección del MPU6050 puede ser 0x68 o 0x69, dependiendo
// del estado de AD0. Si no se especifica, 0x68 estará implicito
```

```
MPU6050 sensor;
// Valores RAW (sin procesar) del acelerometro y giroscopio en los ejes x,y,z
int ax, ay, az;
int gx, gy, gz;
//Variables usadas por el filtro pasa bajos
long f_ax,f_ay, f_az;
int p_ax, p_ay, p_az;
long f_gx,f_gy, f_gz;
int p_gx, p_gy, p_gz;
int counter=0;
//Valor de los offsets
int ax_o,ay_o,az_o;
int gx_o,gy_o,gz_o;
```

Luego, en el *setup*, se inicia al puerto serial, la comunicación con el MPU-6050, y al MPU-6050, así como se inician las lecturas de los componentes “x, y, z”, para luego mostrar un mensaje que muestre estos valores, y donde también se hace la invitación a iniciar la calibración del MPU-6050:

```
void setup()
{
  Serial.begin(57600);   //Iniciando puerto serial
  Wire.begin();          //Iniciando I2C
  sensor.initialize();    //Iniciando el sensor
  if (sensor.testConnection()) Serial.println("Sensor iniciado correctamente");
  // Leer los offset los offsets anteriores
  ax_o=sensor.getXAccelOffset();
  ay_o=sensor.getYAccelOffset();
  az_o=sensor.getZAccelOffset();
  gx_o=sensor.getXGyroOffset();
  gy_o=sensor.getYGyroOffset();
  gz_o=sensor.getZGyroOffset();
  Serial.println("Offsets:");
  Serial.print(ax_o); Serial.print("\t");
  Serial.print(ay_o); Serial.print("\t");
  Serial.print(az_o); Serial.print("\t");
  Serial.print(gx_o); Serial.print("\t");
  Serial.print(gy_o); Serial.print("\t");
```

```
  Serial.print(gz_o); Serial.print("\t");
  Serial.println("nnEnvie cualquier caracter para empezar la calibracionnn");
  // Espera un caracter para empezar a calibrar
  while (true){if (Serial.available()) break;}
  Serial.println("Calibrando, no mover IMU");
}
```

Por último, en el *loop*, es donde se lleva a cabo la calibración del MPU-6050, empezando por leer la velocidad angular y la aceleración de cada componente X, Y, Z. Luego, filtra las lecturas, para que cada cien lecturas puedan corregir los valores que arroja el sensor, y con esto concluir la calibración del mismo:

```
void loop()
{
  // Leer las aceleraciones y velocidades angulares
  sensor.getAcceleration(&ax, &ay, &az);
  sensor.getRotation(&gx, &gy, &gz);
  // Filtrar las lecturas
  f_ax = f_ax-(f_ax>>5)+ax;
  p_ax = f_ax>>5;
  f_ay = f_ay-(f_ay>>5)+ay;
  p_ay = f_ay>>5;
  f_az = f_az-(f_az>>5)+az;
  p_az = f_az>>5;
  f_gx = f_gx-(f_gx>>3)+gx;
  p_gx = f_gx>>3;
  f_gy = f_gy-(f_gy>>3)+gy;
  p_gy = f_gy>>3;
  f_gz = f_gz-(f_gz>>3)+gz;
  p_gz = f_gz>>3;
  //Cada 100 lecturas corregir el offset
  if (counter==100){
    //Mostrar las lecturas separadas por un [tab]
    Serial.print("promedio:"); Serial.print("t");
    Serial.print(p_ax); Serial.print("\t");
    Serial.print(p_ay); Serial.print("\t");
    Serial.print(p_az); Serial.print("\t");
    Serial.print(p_gx); Serial.print("\t");
```

```
   Serial.print(p_gy); Serial.print("\t");
   Serial.println(p_gz);
   //Calibrar el acelerometro a 1g en el eje z (ajustar el offset)
   if (p_ax>0) ax_o--;
   else {ax_o++;}
   if (p_ay>0) ay_o--;
   else {ay_o++;}
   if (p_az-16384>0) az_o--;
   else {az_o++;}
   sensor.setXAccelOffset(ax_o);
   sensor.setYAccelOffset(ay_o);
   sensor.setZAccelOffset(az_o);
   //Calibrar el giroscopio a 0º/s en todos los ejes (ajustar el offset)
   if (p_gx>0) gx_o--;
   else {gx_o++;}
   if (p_gy>0) gy_o--;
   else {gy_o++;}
   if (p_gz>0) gz_o--;
   else {gz_o++;}
   sensor.setXGyroOffset(gx_o);
   sensor.setYGyroOffset(gy_o);
   sensor.setZGyroOffset(gz_o);
   counter=0;
  }
 counter++;
}
```

Con esto, se obtuvieron los datos mostrados en la figura 4.5, con los cuales, se hizo una escala para determinar el valor máximo y mínimo para cada los valores de “x, y, z”.

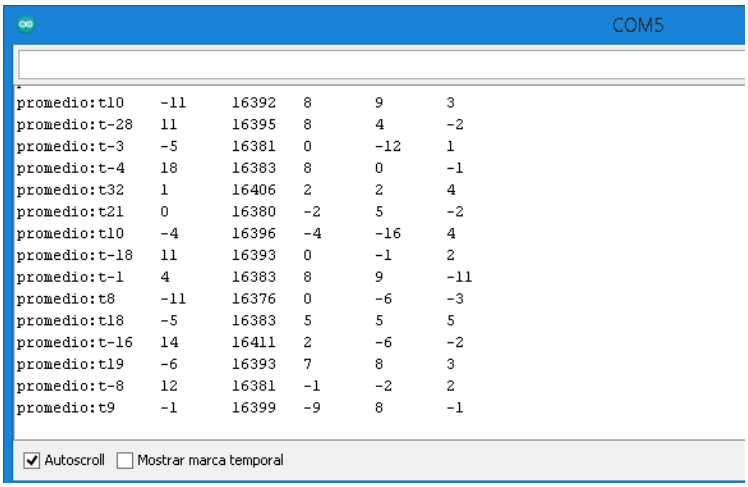

Figura 4.5 Valores presentes en los componentes espaciales del MPU6050.

Una vez obtenida la escala, lo siguiente fue comprobar qué tipo de tareas se podía llevar a cabo con los datos obtenidos en la prueba anterior. Para esto, primero se realizó un filtro en los datos. Esta idea fue tomada de Jonathan Bazán[1], y consiste en un código que aplica un tipo de filtrado a los datos provenientes del acelerómetro. Primero, se utiliza la biblioteca “MPU6050”, así como distintas variables con las que se podrá llevar a cabo la tarea de filtrar la información, y otras más para encender unos LED que indicarán hacia dónde se está inclinando el MPU-6050:

```
#include <MPU6050.h>  //Biblioteca del acelerómetro.
MPU6050 mpu;  //Declaración del sensor.
int16_t ax, ay, az, gx, gy, gz; //Variables del sensor.
int adelante = 2;
int izquierda = 4;
int atras = 5;
int derecha = 3;
//*****************************************************//
const int numLecturas = 15; //Número de muestras para promedio.
int lecturas[numLecturas];  //Lecturas de la entrada analógica.
int indice = 0; //El índice de la lectura actual.
int total = 0;  //Total.
float promedioX = 0;  //Promedio.
```

[1] https://www.youtube.com/watch?v=O7GcR_42rmk&t=227s

En el *setup*, se inicia al monitor serial, al MPU-6050, y se definen los pines que serán utilizados para los LED. También se hace un testeo de que el sensor se encuentra conectado y funcionando:

```
void setup()
{
  Serial.begin(9600);
  //Se inician todas las lecturas a la entrada cero.
  for (int lecturaA = 0; lecturaA < numLecturas; lecturaA++)
    lecturas[lecturaA] = 0;
  for (int lecturaB = 0; lecturaB < numLecturas2; lecturaB++)
    lecturas2[lecturaB] = 0;
  for (int lecturaC = 0; lecturaC < numLecturas3; lecturaC++)
    lecturas3[lecturaC] = 0;
  mpu.initialize(); //Se inicia al acelerómetro.
  pinMode(adelante, OUTPUT);
  pinMode(izquierda, OUTPUT);
  pinMode(atras, OUTPUT);
  pinMode(derecha, OUTPUT);
  if (!mpu.testConnection()){
    while (1);
  }
}
```

Ya en la función *loop*, se realiza el código del filtro, el cual tiene la intención de reducir el valor de los datos entregados por el acelerómetro. Esto es, en lugar de que se trabaje con valores de 15000 a -15000 (valores obtenidos si el acelerómetro se calibra sin fallas), se trabaja con valores de 15 a -15. Al cambiar la escala en la información, realizar un mapeo de la misma resulta en valores más sencillos para utilizar. Ya con los datos filtrados de la forma deseada, se pasa a realizar el encendido del LED de acuerdo a la posición del acelerómetro. Al obtener valores entre 15 y -15 para cada componente X, Y, Z, se escogió un valor de 5 y -5 en los valores de “x” y “y” para encender el LED. Así, se conectaron cuatro LED a las salidas digitales del Arduino Nano (Figura 4.6) en donde se aprecian cinco estados: Uno por cada led prendido, y uno extra donde los cuatro LED están prendidos, indicando que se encuentra en equilibrio. En la figura 4.7 se aprecian los valores de

las componentes espaciales, y cómo los LED encienden de acuerdo a estos datos, mientras que el código es el siguiente:

```
mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz); //Se obtienen variables del senor.
  ax = -(ax / 1000);  //Mapear monitor adelante-atrás.
  ay = (ay / 1000); //Mapear monitor izquierda-derecha.
  az = (az / 1000); //Mapear monitor arriba-abajo.
  //******** Filtro eje X ********//
  total = total - lecturas[indice]; //Se resta la última lectura.
  lecturas[indice] = ax; //Lecturas del sensor.
  total = total + lecturas[indice]; //Se añade la lectura al total.
  indice = indice + 1;  //Se avanza a la próxima posición del array.
  // Cuando se alcanza el final del array.
  if (indice >= numLecturas)
    indice = 0; //Se vuelve al inicio.
  //Se calcula el promedio.
  promedioX = total / numLecturas;  //Se manda a la PC como un valor ASCII.
//******** Filtro eje Y ********//
  .
  .
  .
  Serial.print(promedioX);
  Serial.print(",");
  Serial.print(promedioY);
  Serial.print(",");
  Serial.println(promedioZ);
  //** "X" **//
  if (promedioX <= -5)
  {
    digitalWrite(adelante, LOW);
    digitalWrite(atras, LOW);
    digitalWrite(izquierda, HIGH);
    digitalWrite(derecha, LOW);
    delay(5);
  }
  if (promedioX >= 5)
  {
```

```
  digitalWrite(adelante, LOW);
  digitalWrite(atras, LOW);
  digitalWrite(izquierda, LOW);
  digitalWrite(derecha, HIGH);
  delay(5);
}
//** "Y" **//
if (promedioY <= -5)
{
  digitalWrite(adelante, LOW);
  digitalWrite(atras, HIGH);
  digitalWrite(izquierda, LOW);
  digitalWrite(derecha, LOW);
  delay(5);
}
if (promedioY >= 5)
{
  digitalWrite(adelante, HIGH);
  digitalWrite(atras, LOW);
  digitalWrite(izquierda, LOW);
  digitalWrite(derecha, LOW);
  delay(5);
}
//** Equilibrio **//
if (promedioX >= -4 && promedioX <=4 && promedioY >=-4 && promedioY <= 4)
{
  digitalWrite(adelante, HIGH);
  digitalWrite(atras, HIGH);
  digitalWrite(izquierda, HIGH);
  digitalWrite(derecha, HIGH);
  delay(5);
}
```

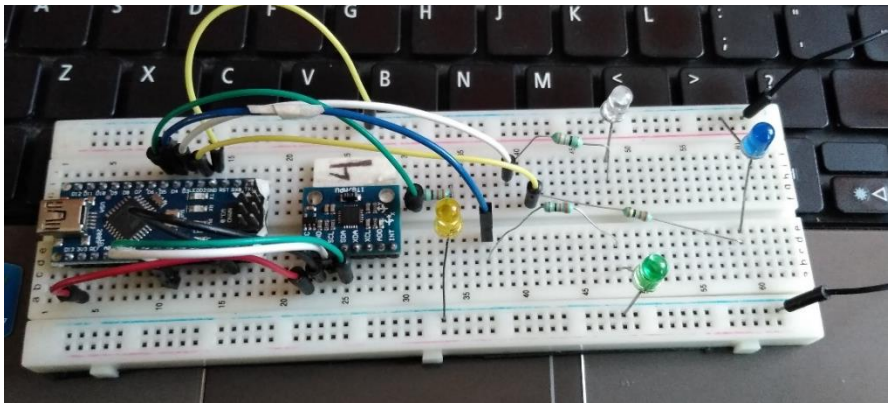

Figura 4.6 Circuito utilizado para el experimento de encendido de varios LED.

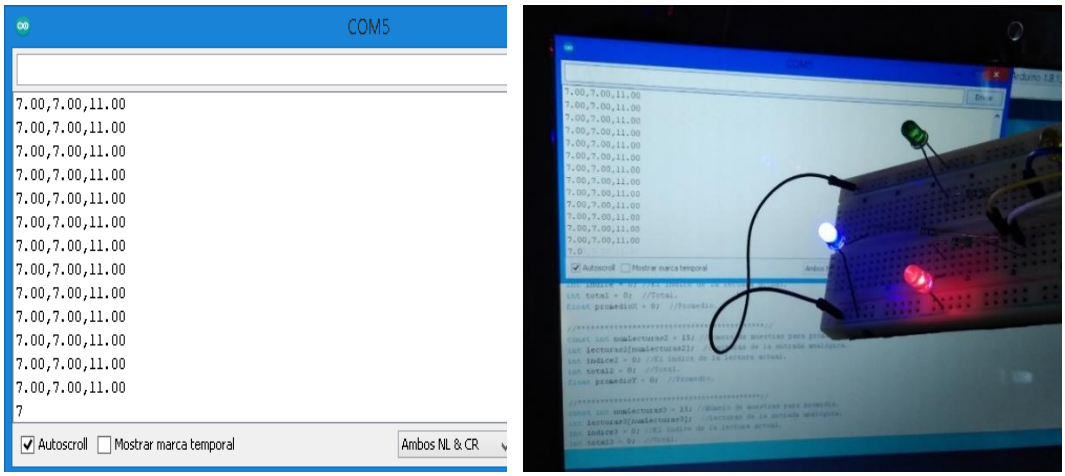

Figura 4.7 Muestreo de valores entregados por el MPU6050 (izquierda), y LED actuando a partir de esta información (derecha).

Como se puede apreciar en las imágenes en conjunto, los leds correspondientes a “izquierda” y “adelante” (pines digitales 2 y 3) se encuentran encendidos, lo que corresponde a los datos mostrados en el monitor serial.

4.3 Medidor de voltaje

Se realizó una serie de experimentos para poder obtener el nivel de batería con el que se energiza a cada módulo esclavo. Los experimentos se muestran a continuación.

Primero, había que saber cómo es que la tarjeta Arduino puede digitalizar valores como el de cuánta cantidad de voltaje tiene una pila. Esto se logró utilizando el convertidor analógico-digital con el que cuenta Arduino. En la tarjeta Arduino Nano, el pin A0 sirve para utilizar este convertidor, por lo que sólo se le conecta lo que se desea digitalizar. Así, se elaboró el circuito de la figura 4.8, en donde hay un

potenciómetro de 5 kOhm conectado a la salida de 5 V de Arduino, y del mismo potenciómetro sale el valor de voltaje que queremos que Arduino cuantifique. El fin de utilizar el potenciómetro es para que funcione como un divisor de voltaje, y con esto, se pueda obtener la escala de valores de voltaje de 5V a 0V sin tener que esperar a que se descargue la pila por conectarle una carga.

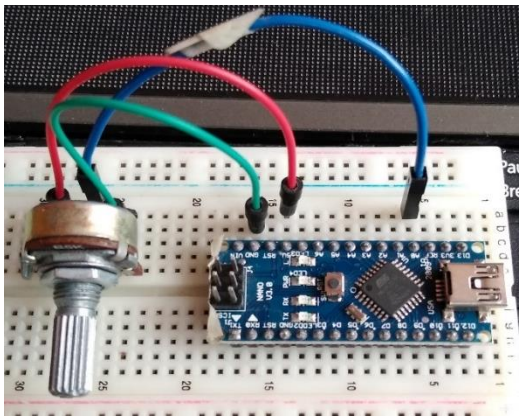

Figura 4.8 Circuito utilizado para las pruebas del convertidor analógico-digital.

Lo siguiente fue obtener el programa que permitiera esta digitalización. Recurriendo a (Arduino, 2021), se obtuvo un programa en el que primero se hace la declaración de las variables “pinAna”, “Vol” y “voltaje”, la primera de tipo *const int*, y las otras dos de tipo *float*. “pinAna” se inicia con valor de A0, “Vol” y “voltaje” se inician con cero. En el *setup* sólo se inicia el monitor serial, con el fin de poder visualizar los valores que posteriormente se le pedirán al programa. En el *loop*, lo primero es que la variable “Vol” convierta el valor de voltaje que está entrando por el pin A0. Luego, a través de una fórmula obtenida por una “regla de tres” (en donde, si 5V=1023 bits de definición, entonces Voltaje= valor correspondiente del convertidor analógico-digital), se le asigna a la variable “voltaje” el nivel de voltaje que está entrando por el pin A0, usando el valor guardado en la variable “Vol” en dicha fórmula. Después, se hace una impresión de estos valores en el monitor serial. La figura 4.9 muestra los valores obtenidos en el monitor serial, mientras que el código es el siguiente:

```
const int pinAna = A0;
float Vol = 0, voltaje = 0;

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

```
}

void loop() {
  Vol = analogRead(pinAna);
  voltaje = (Vol * 5)/1023;
  Serial.print("Valor analógico = ");
  Serial.println(Vol);
  Serial.print("Voltaje = ");
  Serial.println(voltaje);
  delay(250);
}
```

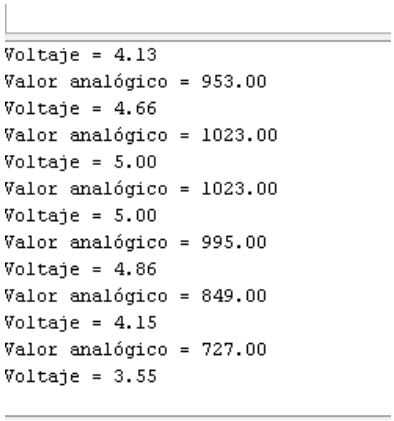

Figura 4.9 Impresión de datos obtenidos en la prueba de medición de voltaje.

Una vez obtenido el método para cuantificar el nivel de voltaje, lo siguiente era ver si se podía obtener el valor de voltaje de una batería o fuente de alimentación externa a Arduino. Para esto, se realizó un código, el cual, hace prender unos LED dependiendo del nivel de voltaje que entre por el pin analógico A0 de Arduino Nano. Para hacer variar de forma rápida el voltaje que recibe el pin analógico, se volvió a utilizar el potenciómetro de 5 kOhm como divisor de voltaje. El circuito empleado en este experimento se aprecia en la figura 4.10.

El código que se empleó inicia con una serie de variables necesarias para cuantificar el nivel de voltaje presente en el divisor de voltaje y para activar los LED empleados para visualizar dicha cuantificación. En el *setup*, se inicia al puerto serial, y se declaran los pines por los que se conectarán a los LED. El *loop* contiene una subrutina llamada “Nivel”, que es en la que se hace la medición de voltaje que está entrando por uno de los pines analógicos del Arduino Nano (previamente

seleccionado en el *setup*), para luego pasar a una serie de condicionales *if*, en donde se asigna

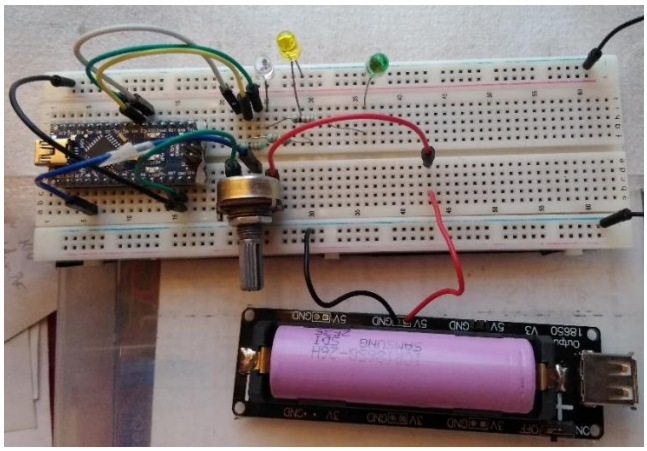

Figura 4.10 Circuito utilizado para las pruebas de obtención del nivel de voltaje en la batería.

Nótese que el nivel de voltaje que se está digitalizando proviene de una salida de 5V ubicada en la base de la batería. Al ser la batería de litio, la forma en la que se va reduciendo la cantidad de voltaje con la que cuenta es gradual, para después caer a cero de manera súbita. Por lo que lo siguiente fue cuantificar el nivel de la batería directamente de sus terminales, así como obtener un estimado de cuánto voltaje puede seguir suministrando antes de bajar de súbito a un valor cercano a 0V.

Para esto, lo primero que se hizo fue cargar la batería a su máxima capacidad. Cabe mencionar que, para cada batería, el voltaje con el que cuentan cuando están cargadas al máximo, varía un poco una de otra. Más adelante se muestran estos valores obtenidos para cada batería. Una vez cargada la batería, se procedió a conectarla a un circuito para que se fuera descargando, y así medir en cuánto tiempo pasa de su máxima capacidad de voltaje a una capacidad cercana a 0V. El circuito utilizado para estas pruebas se muestra en la figura 4.11, en el que se puede apreciar un conjunto de LED con sus respectivas resistencias, así como dos motorreductores, y el circuito es alimentado directamente de la batería, pese a que esta se encuentra colocada en la base utilizada para cargarla. Ya que el fin de este circuito era el de consumir el voltaje de la batería, se fue adecuando para poder hacerlo en un tiempo relativamente corto.

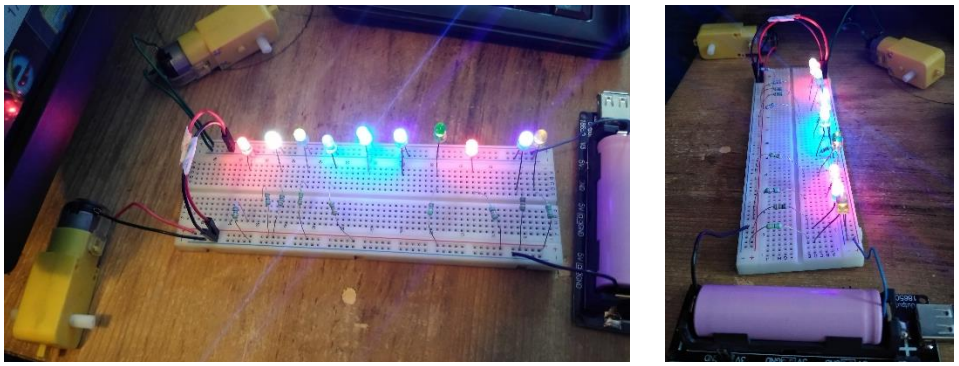

Figura 4.11 Circuito utilizado para las pruebas de capacidad de la batería.

Una vez descargada la batería, se volvió a cargar a su máxima capacidad, y se volvió a conectar al mismo circuito para descargarla, pero esta vez, dentro del tiempo que le tomó descargarse la primera vez, se fue haciendo una medición del voltaje con el que contaba la batería en un determinado tiempo. Esto se hizo así porque se quiere mostrar en la aplicación cuatro estados para el nivel de batería: Nivel alto, nivel medio, nivel bajo y nivel mínimo; en este último, se hará el aviso de que debe cargarse la batería. Como la batería tardó aproximadamente seis horas en descargarse la primera vez, se dividieron estas seis horas en cuatro lapsos de tiempo, para poder obtener el nivel de voltaje en cada uno de estos lapsos. Esto es, se tomaron en cuenta los valores obtenidos por las mediciones hechas cada hora con treinta minutos (01:30) del nivel de voltaje presente en la batería.

El propósito de esta prueba era obtener el lumbral de voltaje con el que la batería puede continuar alimentando un circuito sin problema, además de que el voltaje que se va a tener en consideración para los niveles que mostrará la aplicación será precisamente el de la batería, y no del regulador con el que cuenta la base en la que se carga la batería.

Al tener los niveles de voltaje para los intervalos de tiempo, se obtuvo la gráfica que se muestra en la figura 4.12, en la que se aprecia cómo el voltaje en la batería disminuye de forma gradual y, cerca de su tiempo final de duración, el voltaje en la misma baja de forma súbita.

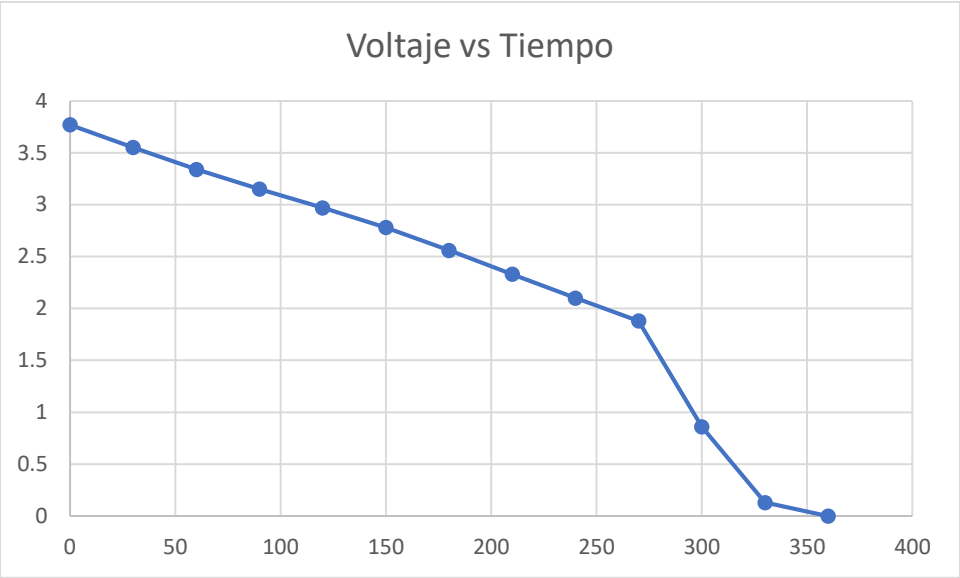

Figura 4.12 Gráfica de Voltaje/Tiempo para una de las baterías.

Con estos datos, se formuló una ecuación para obtener la cantidad de bits que le corresponderían en el programa a esos niveles de voltaje, con el fin de las mediciones tomadas a lo largo de las seis horas coincidiera con lo que cuantificará el Arduino Nano. Esta ecuación es:

$$V = \frac{3*1024}{3.7} \qquad \text{(Ec. 3)}$$

Esta fórmula se obtiene del valor de voltaje mínimo con el que el Arduino Nano y la antena radiofrecuencia pueden operar sin apagarse, que es el “3”. Los “1024” son la cantidad total de bits que ocupa el convertidor analógico/digital del Arduino Nano. El “3.7” es el voltaje máximo que puede tener la batería.

Con el valor de voltaje mínimo con el que pueden operar la antena radiofrecuencia y el Arduino Nano, se usó el convertidor analógico/digital del Arduino, y se incluyó la ecuación previamente mostrada en el código, con el fin de conocer qué valor de bis le corresponde un valor de 3 V.

4.4 Zumbador

El código utilizado para esta prueba inicia declarando la variable “buz” con valor de ‘3’, y esta variable es de tipo *const int*. En el *setup*, únicamente se declara al pin por el que saldrá la señal para que suene el zumbador, y en el *loop*, se realiza la

instrucción de hacer que el zumbador suene por un periodo de 250 milisegundos, luego esperar tres segundos, y luego, apagar el zumbador, como se muestra a continuación:

```
const int buz = 3;

void setup() {
  pinMode(buz, OUTPUT);
}

void loop() {
  tone(buz, 3250, 250);
  delay(3000);
  noTone(buz);
  //analogWrite(buz, 150);
}
```

En un principio, únicamente se había conectado al zumbador a la salida digital del Arduino Nano, ubicada en el pin D3. Pero al hacer esto, el sonido que emitía el zumbador era tan alto que resultaba molesto, por lo que se optó por adicionar un transistor BC547B al circuito, obteniendo el circuito mostrado en la figura 4.13.

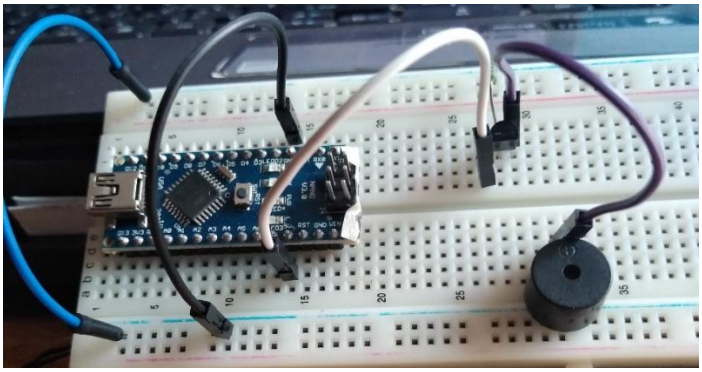

Figura 4.13 Circuito utilizado para las pruebas con el zumbador.

4.5 Antena Bluetooth y aplicación

En esta parte, se inició con las pruebas para la antena Bluetooth. Lo primero fue verificar que la antena funcionaba de manera correcta, por lo que se buscó un código que permitiera verificar esto. Dicho código, obtenido de (Xukyo, 2018), permite ingresar al modo de configuración por comando AT para la antena

Bluetooth. Si bien sólo se buscaba verificar el funcionamiento de la antena Bluetooth, se aprovechó esta oportunidad de poder configurar a la antena con los comandos AT, por lo que se le cambió el nombre y la contraseña para poder conectarse a ella, siendo estos “AntenaBt“ y “1234“, respectivamente.

El código utilizado en esta prueba inicia con la biblioteca “SoftwareSerial”, que sirve para poder conectar la antena Bluetooth en cualquier par de pines del Arduino que no sean los de comunicación serial. También, se definen los pines que se quieren utilizar para conectar la antena, así como la tasa de baudios a la que va a operar la antena:

```
#include <SoftwareSerial.h>
#define rxPin 62
#define txPin 63
#define baudrate 38400
String msg;
SoftwareSerial HC05(rxPin, txPin);
```

Luego, en el *setup*, se declaran los pines para la antena Bluetooth como entrada y salida, que servirán para las terminales “Rx” y “Tx” de la antena, respectivamente. También se inicia al puerto serial, así como a la antena Bluetooth:

```
void setup()
{
  pinMode(rxPin,INPUT);
  pinMode(txPin,OUTPUT);
  Serial.begin(9600);
  Serial.println("Ingrese comandos AT:");
  HC05.begin(baudrate);
}
```

En el *loop*, se invoca a una subrutina llamada “readSerialPort”, para luego entrar a un *if* en el que se comprueba si se recibió alguna instrucción desde el puerto serial. De ser el caso, envía esta instrucción a la antena Bluetooth para que esta la ejecute. Después, con otro *if*, se verifica si hay respuesta de la antena ante dicha instrucción, mostrándola por el puerto serial. La parte de código que hace eso es la siguiente:

```
void loop()
{
```

```
 readSerialPort();
 if(msg!="")
 {HC05.println(msg);}

 if (HC05.available()>0)
 {Serial.write(HC05.read());}
}
```

Para la subrutina, se inicia con una variable de tipo *String*, en la que se almacenará la información recibida por el puerto serial. Para esto, existen dos estructuras de control trabajando en conjunto: la primera es una *while*, de la que no sale hasta haber dejado de recibir información por el puerto serial; la segunda es una *if*, que es justo la que testea si hay datos recibidos por el puerto serial, y los va almacenando en la variabel de tipo *String* previamente declarada. Aquí se muestra lo explicado:

```
void readSerialPort()
{
 msg="";
 while(Serial.available())
 {
  delay(10);
  if (Serial.available() >0 )
  {
   char c = Serial.read();  //gets one byte from serial buffer
   msg += c; //makes the string readString
  }
 }
}
```

El circuito utilizado para esta prueba se muestra en la figura 4.14.

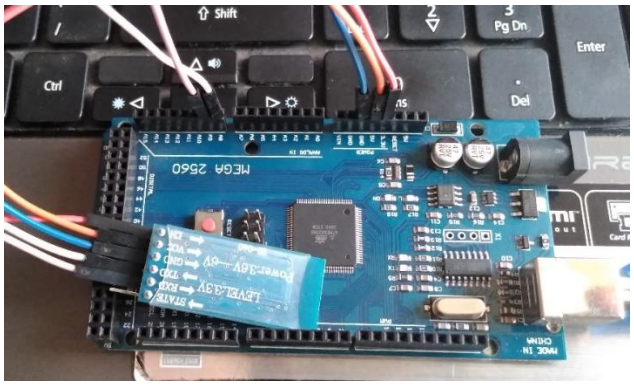

Figura 4.14 Circuito utilizado para las pruebas de configuración de la antena Bluetooth.

Como puede observarse en la figura anterior, el pin *Enable* (o *Key*) de la antena Bluetooth debe estar conectado a 3.3 V. Esto para que el módulo reconozca que lo que se desea es la configuración y no la comunicación como tarea. Además de eso, esta antena cuenta con un botón en el pin *Enable*, el cual debe mantenerse presionado por un par de segundos antes de energizar a la antena para poder establecer el modo de configuración en la antena Bluetooth. La forma de distinguir si la antena está en modo configuración o comunicación es por medio de un LED integrado en la antena, el cual, prende y apaga en intervalos diferentes según sea el caso. Otra cosa a tener en cuenta es que la antena Bluetooth debe estar desconectada de los pines de comunicación; es decir, de los pines “Tx” y “Rx” del Arduino. En la figura 4.15 se muestra el monitor serial de la computadora con algunos de los comandos ingresados para la configuración de la antena Bluetooth.

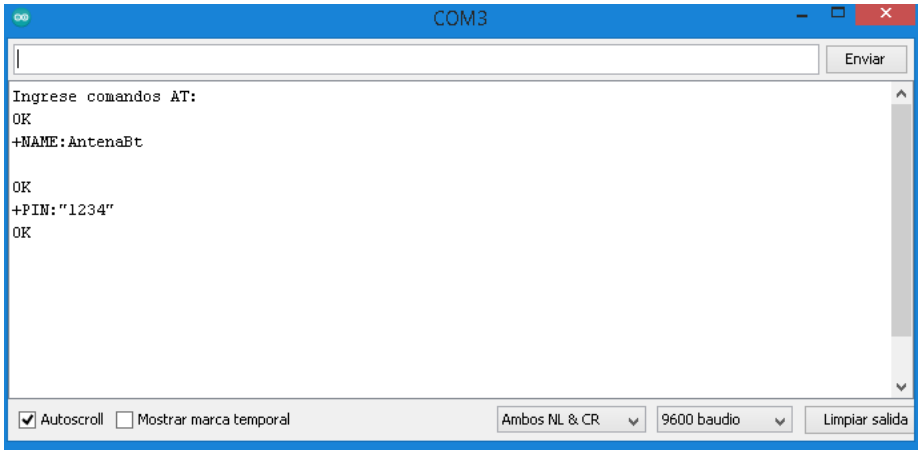

Figura 4.15 Configuración de la antena Bluetooth.

Habiendo logrado configurar a la antena Bluetooth, la siguiente prueba consistió en realizar diferentes tareas a partir de información que se hiciera llegar por el puerto

serial. Esto se logró con el código tomado de (julio, 2020), en el que se hace prender un LED a través de una aplicación para dispositivos Android.

A diferencia de los códigos utilizados hasta este punto, este no incluye bibliotecas. Sólo unas variables que servirán para identificar al pin del LED y los datos recibidos desde la aplicación:

```
//Variables asociadas al LED que se va a controlar
int led_1 = 22;
char valor;  //Variable para indicar que llega una orden
```

Luego, en la función *setup*, se declara como salida al pin en el que se conectará al LED, y se inicia la comunicación serial con una velocidad de 9600 baudios:

```
void setup() {


  pinMode(led_1, OUTPUT);
  Serial.begin(9600);
}
```

En la función *loop*, se inicia con un condicional *if*, que hará la evaluación de si se ha recibido información por parte de la aplicación. En caso de haber, guarda esa información en una variable, para después evaluar el valor de esta variable a través de una serie de condicionales *if*. Ya que sólo se quiere prender y apagar un LED, lo que se va a comparar es si la información que llegó es para encender el LED o para apagarlo:

```
void loop() {
  if (Serial.available())  //Si el puerto serie (Bluetooth) está disponible
  {
    valor = Serial.read();  //Lee el dato entrante via Bluetooth
    if (valor == 'A')   //Si el dato que llega es una A
    {
      digitalWrite(led_1, HIGH);  //Enciende el LED 1
    }
    if (valor == 'B')   //Si el dato que llega es una B
    {
      digitalWrite(led_1, LOW);   //Apaga el LED 1
    }
```

```
  }
}
```

El circuito utilizado para esta prueba se muestra en la figura 4.16.

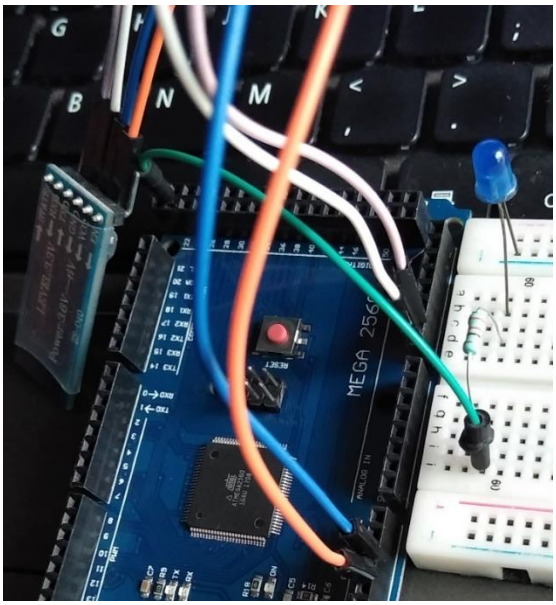

Figura 4.16 Circuito utilizado para las pruebas de comunicación de la antena Bluetooth y la aplicación para dispositivos Android.

Ya que el fin de este experimento era comprender la comunicación entre la antena Bluetooth y la aplicación para dispositivos Android, la estética de la aplicación es bastante sencilla para este experimento, pues la interfaz sólo cuenta con un botón para iniciar la conectividad Bluetooth, y otro para prender y apagar el LED. También, su programación resulta sencilla, ya que se programa al botón de conexión para que inicie y reconozca la conectividad con la antena Bluetooth, y al botón de encendido y apagado del LED para que mande un valor que el Arduino Mega reconoce en su programación. La programación de esta aplicación se muestra en la figura 4.17.

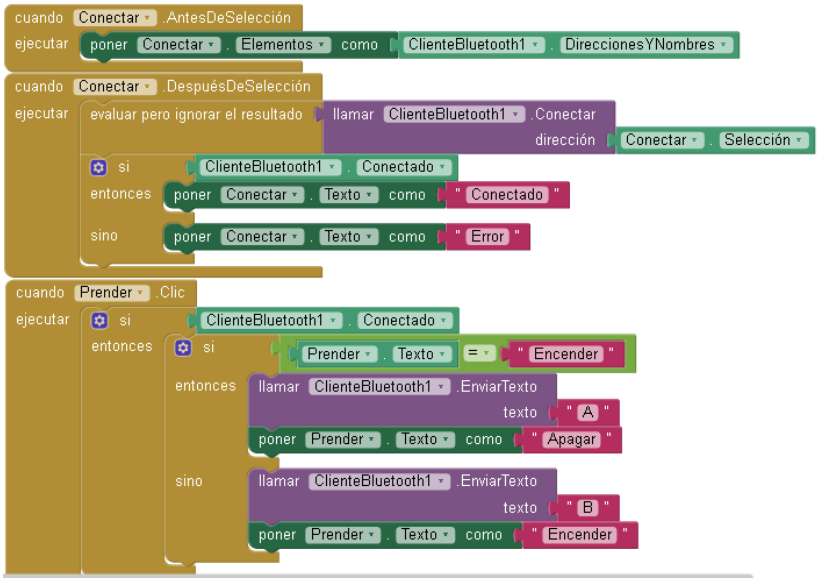

Figura 4.17 Programación de la aplicación utilizada para las pruebas de comunicación con la antena Bluetooth.

Los experimentos para la aplicación se fueron desarrollando con forme se necesitaba incorporar una nueva instrucción a la comunicación entre la antena Bluetooth y la aplicación, entre las cuales está la sincronización entre la velocidad de envío de datos por la antena Bluetooth y la velocidad de muestreo de estos datos en la aplicación para dispositivos Android. También, había que definir una forma de indicar a la aplicación que se ha terminado de enviar información, por lo que se incluyó en el código final una función para indicar el fin del envío de información. Para el envío y recepción de información desde la aplicación, las instrucciones están señaladas por el entorno de desarrollo de AppInventor, y una vez conocida el algoritmo que se debía seguir para lograr enviar y recibir información, se organizaron estas instrucciones para que la aplicación operara de forma correcta.

4.6 Módulos maestro y esclavos funcionando en conjunto

Habiendo terminado las pruebas necesarias para poder llevar a cabo el prototipo propuesto en este escrito, lo siguiente fue aplicar lo aprendido en dichas pruebas.

Iniciando con el módulo maestro, lo que se necesitaba era que se pudiera controlar la iluminación de las tiras LED RGB, la comunicación vía Bluetooth, y la comunicación vía radiofrecuencia. Originalmente, se tuvo en mente que este módulo contara con una pantalla LCD y un reloj externo. La pantalla debía mostrar

el estado de los zumbadores en los módulos esclavos, así como el nivel de batería en los mismos, y la hora e y la fecha con ayuda de un reloj externo. La imagen 4.18 muestra un fragmento de este código.

```
  radio.startListening();//Se incia al módulo RF como receptor.

//Se incia el LCD con el número de filas y columnas, y el mensaje de incio.
  lcd.begin(20,4);
  lcd.clear();
  lcd.setCursor(5,0);
  lcd.print("Bienvenido");
  lcd.setCursor(5,2);
  lcd.print("Grupo IDEA");
  delay(5000);

//Se comprueba la conexión del RTC.
  if(! rtc.begin())
  {
    lcd.clear();
    lcd.println("No hay modulo RTC");
    delay(5000);
    lcd.clear();
    while(1);
  }
//En esta sección se pone a la hora al módulo RTC. Luego, se comenta y se vuelve a cargar. Esto debe reali
//Los valores de fecha y hora son tomados de la fecha y hora en que el sketch fue compilado.
//rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));
  lcd.clear();

  pinMode(9, OUTPUT); // 1X
  pinMode(10, OUTPUT); // 1Y
  pinMode(11, OUTPUT); // 1Z
  pinMode(2, OUTPUT); // 2X
```

Figura 4.18 Fragmento del primer código desarrollado para el módulo maestro.

En el siguiente código desarrollado, se contempla el uso de una aplicación para dispositivos Android para sustituir la tarea visual de la pantalla LCD y la tarea del reloj externo. Además de que la aplicación serviría como control para prender o apagar los zumbadores de los módulos esclavos. Dado que se buscaba enviar dos datos desde los módulos esclavos al módulo maestro (el dato del estado del zumbador y del nivel de batería), este código también incluye la concatenación de información para poder ser enviada por la antena radiofrecuencia. La figura 4.19 muestra una parte de este código.

```
void loop()
{
  if(radio.available()) //Verificando si hay recepción de datos.
  {
    while(radio.available())
    {
      int len = 0;
      char DatoR[6];  //Variable para almacenar los datos recibidos en la antena RF.
      len = radio.getDynamicPayloadSize();  //Descarta el tamaño del paquete de datos.
      radio.read(&DatoR, len);  //Lee el dato recibido y lo guarda en DatoR.

      const char Clavel = ".";  //Caracter que servirá para separar la trama de datos.
      char *Llavel = strtok(DatoR, Clavel); //Identificación de los datos presentes entre separadores.
      strncpy(codigo, Llavel, sizeof(codigo));  //Copiando primer dato separado.
      Llavel = strtok(NULL, Clavel);  //Identificación del siguiente dato presente entre los separadores.
      strncpy(bateria, Llavel, sizeof(bateria));  //Copiando segundo dato separado.
```

Figura 4.19 Fragmento del segundo código desarrollado para el módulo maestro.

En un tercer código, la comunicación bidireccional vía Bluetooth y vía radiofrecuencia ya estaba sentada como idea principal, pero tuvo que ser

modificado porque no se consiguió la comunicación bidireccional vía radiofrecuencia. El módulo maestro no separaba de forma correcta la cadena de datos enviada por los módulos esclavos, y no se hacía el envío de la información que hacía posible prender o apagar los zumbadores de los esclavos, por lo que fue necesario replantear el algoritmo de comunicación de ambos módulos (esclavos y maestro). La figura 4.20 muestra una parte de dicho código.

```
`
  while(radio.available())
  {
    int len = 0;
    char DatoR[6];  //Variable para almacenar los datos recibidos en la antena RF.
    len = radio.getDynamicPayloadSize();  //Descarta el tamaño del paquete de datos.
    radio.read(&DatoR, len);  //Lee el dato recibido y lo guarda en DatoR.
    Serial.println(DatoR);
    const char Clavel = ".";  //Caracter que servirá para separar la trama de datos.
    char *Llavel = strtok(DatoR, Clavel); //Identificación de los datos presentes entre separadores.
    strncpy(codigo, Llavel, sizeof(codigo));  //Copiando primer dato separado.
    Llavel = strtok(NULL, Clavel);  //Identificación del siguiente dato presente.
    strncpy(bateria, Llavel, sizeof(bateria));  //Copiando segundo dato separado.

    C1 = String(codigo).toInt();
    C2 = String(bateria).toInt(); //Convirtiendo los 'char' a 'int' para un uso más sencillo.
    //Serial.print(C1); Serial.print(","); Serial.println(C2);
    asignaColores(C1);  //Subrutina de encendido de las tiras led.
    asignaBateria(C2);  //Subrutina para la identificación del nivel de batería en los esclavos.
  }
}
else{
delay(100);
```

Figura 4.20 Fragmento del tercer código desarrollado para el módulo maestro.

El cuarto código desarrollado fue el definitivo. En este, ya es posible realizar una comunicación bidireccional, tanto vía Bluetooth, como vía radiofrecuencia. Los cambios para este código es que ya no se reciben cadenas de datos, sino que se emplean más casos dentro de una estructura de control *while* en la recepción de datos vía radiofrecuencia, con los que es posible saber tanto la orientación de los módulos esclavos, como el nivel de la batería que los alimenta. La comunicación vía Bluetooth también se lleva a cabo de forma bidireccional, utilizando caracteres simples para el reconocimiento de instrucciones dentro de estructuras de control *if*. Como esto ya se ha explicado en el capítulo tres de este escrito, sólo se hace una breve mención para este código.

El circuito utilizado para el módulo maestro se muestra en la figura 4.21, que fue con el que se llevaron a cabo las pruebas de funcionamiento para el prototipo final.

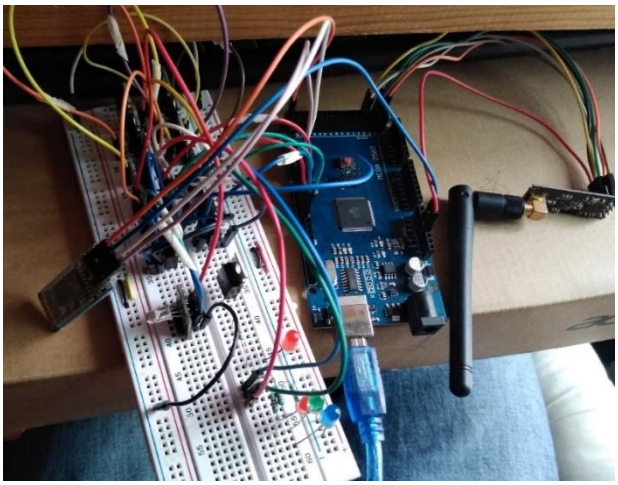

Figura 4.21 Circuito utilizado para las pruebas de funcionamiento del módulo maestro.

Como puede observarse en la figura anterior, en el circuito para las pruebas del módulo maestro no se usa la fuente de poder de 240 W ni las tiras LED RGB. Ya que el primer interés era saber si el maestro recibía y enviaba de forma correcta información, las pruebas para la etapa de potencia se hicieron mientras se probaba la aplicación. Más adelante se mostrarán los resultados de esas pruebas.

La aplicación tuvo modificaciones para conseguir una comunicación estable con la antena Bluetooth del módulo maestro, pero también tuvo modificaciones para cuidar la estética de la misma, pues al principio, contenía colores muy llamativos, que apartaban la atención del objetivo principal de la aplicación, que era controlar. La primera versión de la aplicación se muestra en la figura 4.21.

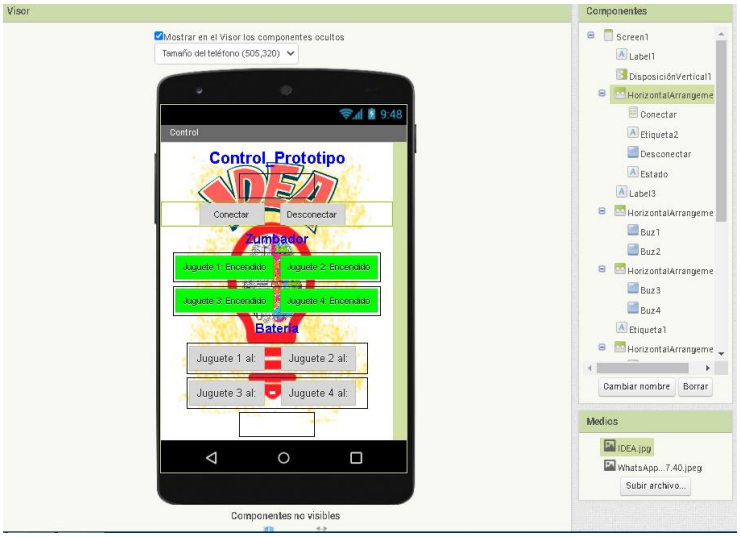

Figura 4.22 Primera versión de la aplicación, empleada para las pruebas de comunicación con el módulo maestro.

Para la siguiente versión de la aplicación, se dispuso de un botón de salida, además de unos cuantos cambios visuales, como se observa en la figura 4.22.

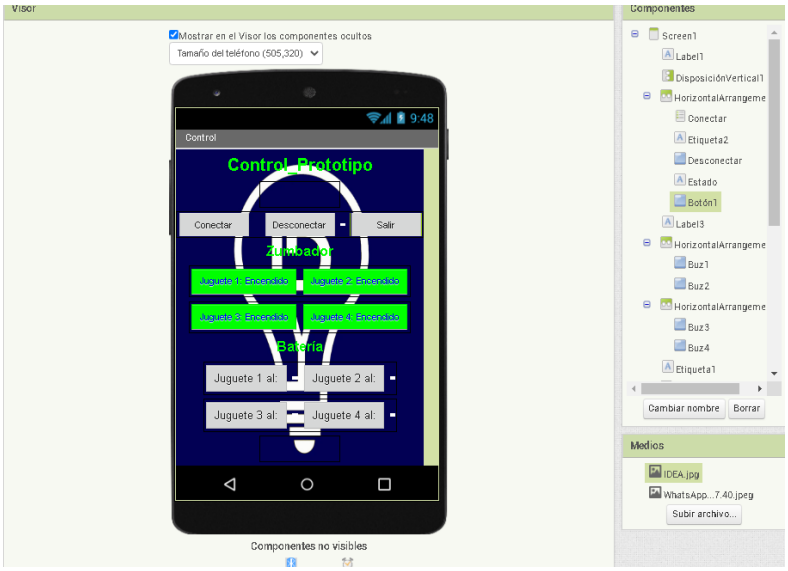

Figura 4.23 Segunda versión de la aplicación.

Para los módulos esclavos, las primeras pruebas se hicieron sólo con dos módulos esclavos. Esto para verificar que no había interferencia entre ellos, así como para comprobar la capacidad del módulo maestro para enviar y recibir información de los esclavos. Al final, se utilizaron los cuatro módulos esclavos propuestos desde un inicio. El primer código obtenido para los esclavos se muestra en la figura 4.24, en la que se puede ver que el código mostraba un error. Este error era que los datos que se querían concatenar para poder ser enviados al módulo maestro no coincidían con el tipo de variable que se había declarado al inicio, y que es en donde se almacenan los datos concatenados. Además, no se conseguía una comunicación bidireccional vía radiofrecuencia con el módulo maestro, por lo que hubo que replantear el algoritmo que permitiría a los módulos comunicarse entre sí. La parte de la obtención de la orientación del módulo esclavo no presentó problemas.

```
      done = radio.write(&Datol, strlen(Datol));//Envio de datos.*/
      Datol = "11.";
      if(Enviado){
        if(Buz) tone(buz, 900, 175);
        else noTone(buz);
      }
      ref = 11;//Se mantiene el estado de 'ref' hasta el siguiente cambio.
      delay(25);
    }
  }
  else
   if(y == 133)  //2 Y
   {
     if(ref!=12)
     {
       //radio.stopListening();
       char Datol[] = "12.";
       strcat(Datol, Dato2);
       radio.openWritingPipe(PTXpipe);//Canal de escritura.
       done = radio.write(&Datol, strlen(Datol));//Envio de datos.
       ref = 12;//Se mantiene el estado de 'ref' hasta el siguiente cambio.
       delay(25);
     }
   }
   else
    if(z == 135)  //3 Z
    {
      if(ref!=13)
      {
       // radio.stopListening();
        char Datol[] = "13.";
        strcat(Datol, Dato2);
        radio.openWritingPipe(PTXpipe);//Canal de escritura.
        done = radio.write(&Datol, strlen(Datol));//Envio de datos.
        ref = 13;//Se mantiene el estado de 'ref' hasta el siguiente cambio.
        delay(25);
      }
    }
    else
```

incompatible types in assignment of 'const char [4]' to 'char [2]'

Copiar mensajes de error

Figura 4.24 Fragmento del primer código desarrollado para los módulos esclavos.

Después, se desarrolló un segundo código. En este, se solucionaba el error de la compatibilidad de datos para concatenar, pero el módulo esclavo no conseguía separarlos de forma correcta, por lo que se optó por otra forma de hacerle saber al módulo maestro la orientación y el nivel de batería de los esclavos. Al mismo tiempo, la comunicación bidireccional ya se hacía, pero luego de un par de intentos para apagar o encender el zumbador, el módulo dejaba de enviar y recibir información. La figura 4.25 muestra una parte del código con el que se obtuvieron los resultados previamente mencionados.

```
  if(ref!=ll)
  {
    radio.stopListening();  //Se pasa al modo de transmisor.
    radio.openWritingPipe(PTXpipe); //Se abre el canal de escritura.
    Datol[6] = "ll."; //Se crea la variable con el dato del código correspondiente.
    strcat(Datol, Dato2); //Se hace la concatenación de datos para ser enviados.
    //Serial.println(Datol);
    if(radio.write(&Datol, strlen(Datol)))  //Envío de datos.
    {
      if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
      else noTone(buz);            //valor retornado en la subrutina del mismo.
    }
    else{
      Serial.println("mal");  //Condición utilizada para pruebas. Sólo visible si se está conectado al P
    }
    ref = ll; //Se mantiene el estado de 'ref' hasta el siguiente cambio.
    delay(25);
    radio.startListening(); //Se regresa al modo de receptor.
  }
}
else
 if(y == 133)  //2 Y
 {
   if(ref!=l2)
   {
```

Figura 4.25 Fragmento del segundo código desarrollado para los módulos esclavos.

Para el tercer y último código, la comunicación bidireccional vía radiofrecuencia se efectuaba de manera correcta y sin interrupciones. Esto se logró cambiando el tipo de dato que se enviaba al módulo maestro. En lugar de concatenar información para luego enviarla, se envía un único caracter, con el que se identifica el nivel de batería y la orientación del módulo esclavo con respecto del suelo. La cuantificación del nivel de voltaje en la batería, así como la orientación del módulo esclavo, se realizan de la misma forma que en el tercer código, pues esas instrucciones se ejecutaron sin problema. De igual forma, se mantienen las instrucciones para encender o apagar el zumbador en el esclavo.

El circuito utilizado para estas pruebas se observa en la figura 4.26.

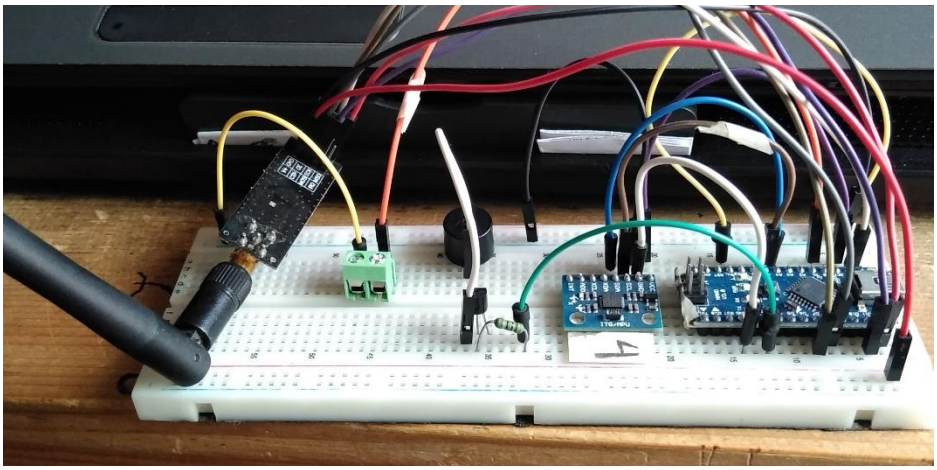

Figura 4.26 Circuito utilizado para las pruebas de funcionamiento de los módulos esclavos.

Para la comprobación de la etapa de potencia, se usó el circuito y las conexiones mostradas en las figuras 4.27 y 2.28, en las que se puede observar que ya se usa la fuente de 240 W para alimentar a las tiras LED. Recordando que, en estas pruebas para los módulos funcionando en conjunto, sólo se usaron dos módulos esclavos y un maestro, por lo que sólo se aprecian dos tiras LED conectadas a la fuente de 240 W y a los transistores IRF640n.

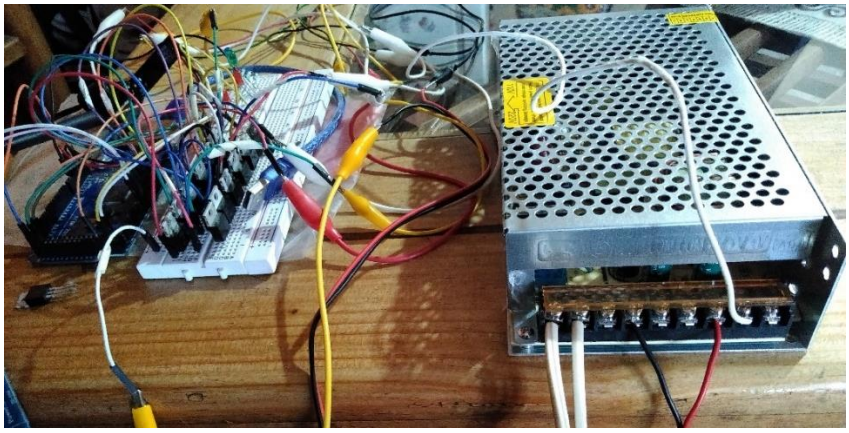

Figura 4.27 Conexiones de la fuente de poder, las tiras LED, los transistores, y Arduino Mega, hechas para realizar las pruebas finales de funcionamiento.

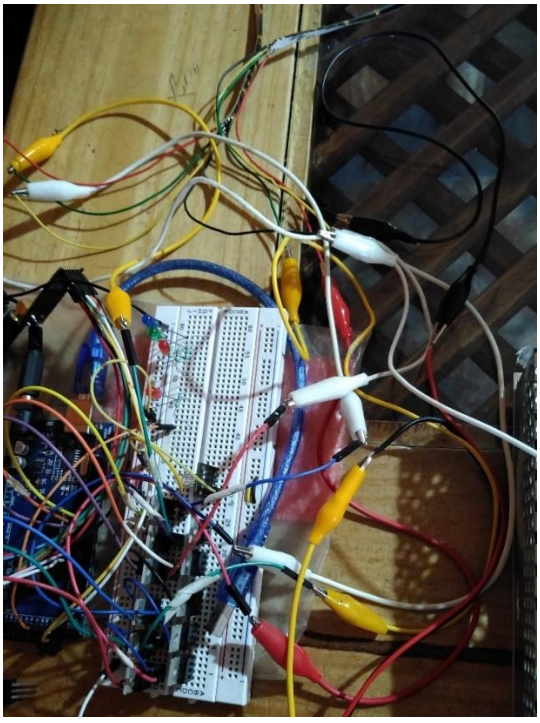

Figura 4.28 Conexiones de Arduino Mega, los transistores y las tiras LED, hechas para las pruebas finales de funcionamiento.

4.7 Elaboración de PCB para los módulos esclavos y maestro.

Como se mencionó en el capítulo tres, el software en el que se diseñaron los planos de las tarjetas PCB fue Eagle. Los primeros modelos, como los que muestran las figuras 4.29 y 4.30, fueron hechos a partir del circuito montado en la protoboard. Este diseño tuvo que ser alterado, pues montar el circuito en la placa de baquelita demandaba piezas que no se habían considerado por la comodidad dada por la protoboard, como lo son algunos *pinhead* y borneras, entre algunos otros.

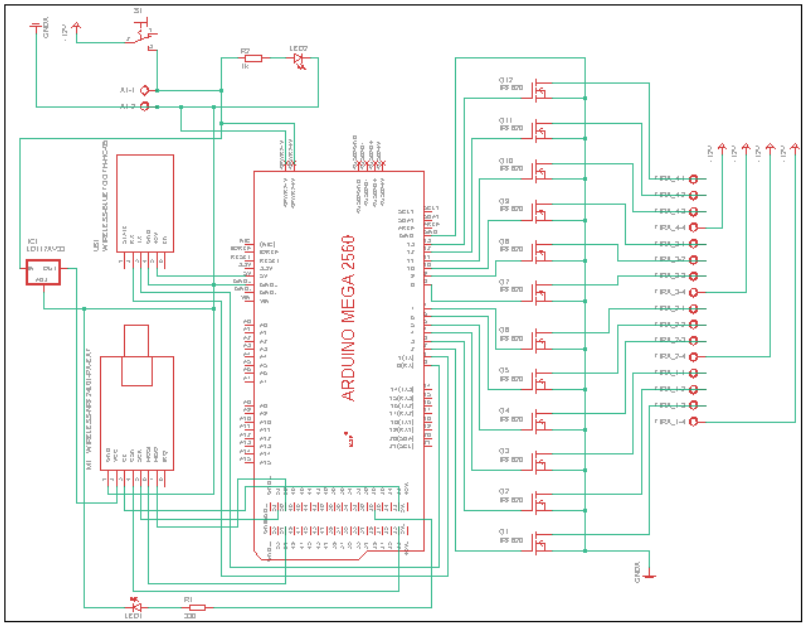

Figura 4.29 Diseño inicial para la PCB del módulo maestro.

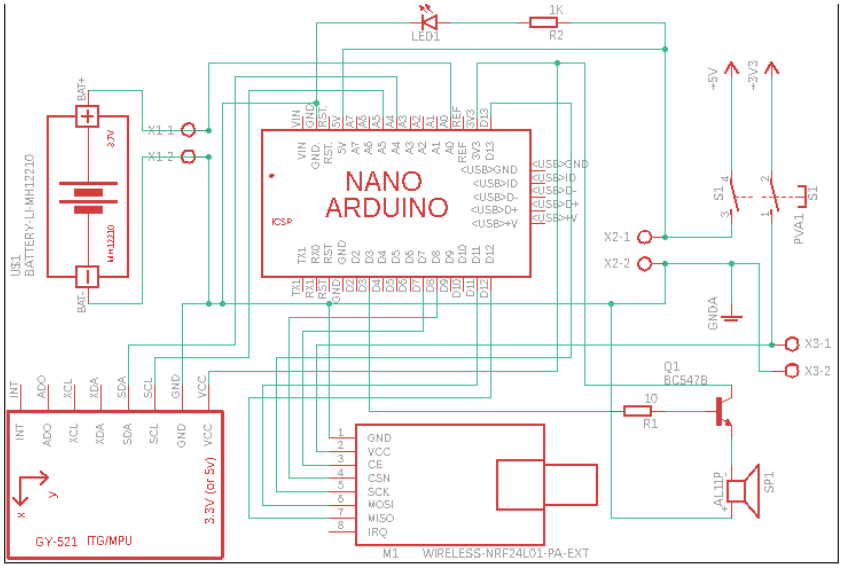

Figura 4.30 Diseño original para la PCB de los módulos esclavos.

Una vez que se tuvo bien en claro qué componentes hacían falta en el circuito, se integraron al diseño final, y se pasó a elaborar la PCB, como se puede observar en la figura 4.29.

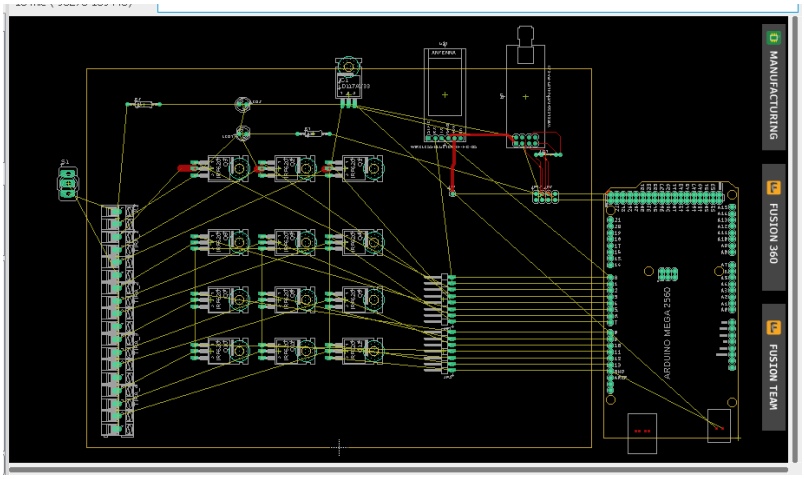

Figura 4.31 Primeros arreglos para la PCB del módulo maestro.

En la parte de realización de las pistas para el módulo maestro, se hicieron con la escala predeterminada de Eagle, pero tuvieron que modificarse, pues el ancho con el que se estaban haciendo las pistas no era el correcto para los componentes. Por ejemplo, como se hace uso de transistores de potencia, se debió hacer las pistas con un ancho que cubriera la demanda de potencia que soportan los transistores.

Para los módulos esclavos, la realización de las pistas se realizaba sin considerar que no puede haber pistas con esquinas cuadradas. Esto es, haciendo un giro de 90 grados en la dirección de las pistas.

Conclusiones

Se logró la incorporación de sonidos distintivos para cada módulo esclavo cuando el usuario cambia la orientación respecto a sus ejes. Para ello, se utilizaron transductores de tipo zumbador, los cuales son económicos y tienen un volumen que permite su escucha de manera adecuada.

Se buscó y analizó información acerca de diversas opciones de acelerómetros/giroscopios, el prototipo inicial incorporaba los de modelo MMA7361, que entregan una señal analógica en función de su orientación. Por esto, se decidió optar por el modelo MPU-6050, los cuales son digitales, y que las pruebas demostraron ser (en ambos casos) eficientes en la funcionalidad requerida. El cambio principalmente fue debido a que su costo es bastante inferior y es más fácil conseguirlos.

Se cambió la etapa de suministro de energía en los módulos esclavos, pasando de un conector para batería, a usar un módulo que incorpora el slot para una batería de litio y una circuitería para la gestión de carga y suministro; adicionalmente, se incorporó un dispositivo que permite la carga inalámbrica, el cual originalmente está creado para teléfonos celulares, esto pensando en que el modelo final deberá estar sellado para ser a prueba de agua, por lo que se considera más práctica la posibilidad de energizar los módulos con este tipo de alternativa.

Se modificó la etapa de potencia para poder alimentar a las tiras LED RGB de tres metros cada una, dado el consumo energético se propuso el empleo de una fuente de 240 W (12V – 20 A), así como transistores de potencia IRF640n, que son transistores con mayor capacidad de potencia que los TIP41c utilizados en el prototipo inicial.

Se diseñó una aplicación para dispositivos Android capaz de permitir encender-apagar los zumbadores en los módulos esclavos, también permite conocer el nivel de carga presente en las baterías que alimentan a cada módulo. A la par, se modificó al módulo maestro para que pudiera conectarse con la aplicación,

haciéndolo mediante Bluetooth, sirviendo como intermediario entre la aplicación desarrollada y los módulos esclavos.

Con esto, se logró aumentar y mejorar las características de un sistema de control de iluminación RGB por posicionamiento en tres ejes. Si bien el prototipo presentado en este escrito aún requiere trabajo para lograr la meta final del proyecto, con lo realizado, se ha logrado un importante acercamiento, recordando que la intensión es su implementación del sistema en el área de estimulación multisensorial del Centro de Rehabilitación e Inclusión Infantil Teletón, del municipio de Nezahualcóyotl.

Trabajo futuro

Como trabajo futuro se tienen consideradas las siguientes actividades:

- Rediseñar las tarjetas PCB para incorporar elementos de montaje superficial y los microcontroladores en lugar de las tarjetas de Arduino.
- Crear carcasas para el módulo maestro, que sean capaces de proteger a la circuitería, así como de mantener a temperatura adecuada mediante la incorporación de ventilación activa/pasiva.
- Crear carcasas para los módulos esclavos, que sean capaces de resistir el uso por parte de los pacientes, así como de ser a prueba de agua, también deben permitir la colocación idónea del receptor de carga inalámbrica.
- Diseñar un módulo que permita el resguardo de los módulos esclavos cuando no estén en operación y que a la vez permitan la recarga de los mismos.
- Diseñar un montaje para que las tiras LED puedan ser colocadas en espacios del CRIIT, tanto la habitación multisensorial, como en la alberca.
- Realizar las acciones de implementación en el CRIIT.

Referencias

Alvarado Alvarado, M. A., & Prado Monge, J. R. (2019). *Universidad del Azuay.* Obtenido de http://dspace.uazuay.edu.ec/handle/datos/8765

Anónimo. (2 de Julio de 2020). *Centros para el Control y la Prevención de las Enfermedades*. Obtenido de https://www.cdc.gov/ncbddd/spanish/cp/facts.html

Arduino. (19 de Noviembre de 2020). Obtenido de https://arduino.cl/que-es-arduino/

Arduino. (20 de Noviembre de 2020). *Arduino Store*. Obtenido de https://store.arduino.cc/usa/mega-2560-r3

Arduino. (29 de Junio de 2021). *Arduino Reference*. Obtenido de file:///C:/Program%20Files%20(x86)/Arduino/reference/www.arduino.cc/en/Reference/AnalogRead.html

Arduino. (11 de 09 de 2021). *Arduino Store*. Obtenido de https://store-usa.arduino.cc/products/arduino-nano?selectedStore=us

Arduino. (27 de Abril de 2021). *ArduinoReference.* Obtenido de https://www.arduino.cc/en/reference/SPI

Area Tecnología. (22 de Noviembre de 2020). *Tecnología*. Obtenido de https://www.areatecnologia.com/TUTORIALES/EL%20TRANSISTOR.htm

Balsells, R. (15 de Noviembre de 2017). *Rosa Ma. Balsells Penas*. Obtenido de http://psicologostortosa.com/beneficios-la-estimulacion-sensorial/

Biblioteca Nacional de Medicina de los EE. UU. (15 de Julio de 2020). Obtenido de https://medlineplus.gov/spanish/downsyndrome.html

Biblioteca Nacional de Medicina de los EE. UU. (30 de Diciembre de 2020). *MedlinePlus*. Obtenido de https://medlineplus.gov/spanish/birthdefects.html

Casillas, R., L. Morán, A., & Meza-Kubo, V. (2018). Obtenido de http://fc.ens.uabc.mx/documentos/fica/resumenes/Casillas_ResumenFICA_2018-vF.pdf

Cid Rodríguez, M. J., & Camps Llauradó, M. (20 de Septiembre de 2010). Obtenido de https://sid.usal.es/idocs/F8/ART18862/236-2%20Cid.pdf

Fisiolab. (3 de Noviembre de 2020). Obtenido de https://fisiolab.mx/fisiolab/wp-content/uploads/2018/02/CEMS-FISIOLAB.pdf

García Gonzáles, A. (4 de Enero de 2016). *PanamaHitek*. Obtenido de http://panamahitek.com/que-es-y-como-funciona-un-mosfet/

Gómez, E. (2010). *RinconIngenieril.* Obtenido de https://www.rinconingenieril.es/que-es-pwm-y-para-que-sirve/

Gonzalez Gonzalez, S. R., & Guachun Arias, J. C. (Junio de 2018). *Universidad Politécnica Salesiana.* Obtenido de http://dspace.ups.edu.ec/handle/123456789/15727

Google Sites. (16 de Noviembre de 2020). *Modelo de color RGB*. Obtenido de https://sites.google.com/site/combinaciondecolores/home/modelo-de-color-rgb

Grupo TME. (9 de Octubre de 2020). Obtenido de https://www.tme.com/mx/es/news/library-articles/page/22568/Como-funciona-y-que-hace-el-acelerometro/

Hisour. (12 de Noviembre de 2020). Obtenido de https://www.hisour.com/es/rgb-color-model-24867/

International Rectifier. (24 de Marzo de 2021). *AllDataSheet.* Obtenido de https://pdf1.alldatasheet.com/datasheet-pdf/view/68219/IRF/IRF640N.html

InvenSense Inc. (16 de Mayo de 2012). *AllDataSheet.* Obtenido de https://pdf1.alldatasheet.com/datasheet-pdf/view/517744/ETC1/MPU-6050.html

Isaac. (13 de Noviembre de 2020). *HardwareLibre*. Obtenido de https://www.hwlibre.com/led-rgb/

jecrespom. (13 de Marzo de 2021). *Aprendiendo Arduino*. Obtenido de https://aprendiendoarduino.wordpress.com/2016/11/13/bluetooth-en-arduino/

julio, e. (17 de 03 de 2020). *Arduino para todos*. Obtenido de http://arduparatodos.blogspot.com/2017/06/modulo-bluetooth-hc-06-con-arduino-y.html

Kohen, V. H. (21 de Septiembre de 2021). *iluminet*. Obtenido de https://www.iluminet.com/tiras-led/

Leds International. (13 de Noviembre de 2020). Obtenido de https://www.ledsinternational.com/es/que-son-los-leds/index.html

Llamas, L. (8 de Diciembre de 2016). Obtenido de https://www.luisllamas.es/comunicacion-inalambrica-a-2-4ghz-con-arduino-y-nrf24l01/

Llamas, L. (14 de Mayo de 2016). *Luis Llamas.* Obtenido de Ingeniería, informática y diseño: https://www.luisllamas.es/arduino-spi/

Madriaga, R. (08 de Septiembre de 2015). *Curso Arduino Mega*. Obtenido de http://cursoarduinomega.blogspot.com/2015/09/hc-06-bluetooth-module-slave-with.html

Mateos Peña, F. C. (Febrero de 2018). Sistema controlador de iluminación RGB por posicionamiento en 3 ejes como estímulo multisensorial para el área de hidroterapia del Centro de Rehabilitación e Inclusión Infantil Teletón, Nezahualcóyotl. Estado de México, México.

Millervet. (15 de Junio de 2021). *Instructables Circuits*. Obtenido de https://www.instructables.com/NRF24L01-Multiceiver-Network/

Molina Velázquez, T., & Banguero Millán, L. F. (2008). Diseño de un espacio sensorial para la estimulación temprana de niños con multidéficit. *Revista Ingeniería Biomédica*, 40-47.

Naylamp Mechatronics. (16 de Noviembre de 2020). Obtenido de https://www.naylampmechatronics.com/blog/45_Tutorial-MPU6050-Aceler%C3%B3metro-y-Giroscopio.html

Naylamp Mechatronics. (22 de Marzo de 2021). *NAYLAMP MECHATRONICS*. Obtenido de https://naylampmechatronics.com/inalambrico/43-modulo-bluetooth-hc05.html

Nemours. (Febrero de 2017). *TeensHealt.* Obtenido de https://kidshealth.org/es/teens/pt-esp.html

NORDIC SEMICONDUCTOR. (Marzo de 2008). *Sparkfun.* Obtenido de https://www.sparkfun.com/datasheets/Components/SMD/nRF24L01Pluss_Preliminary_Product_Specification_v1_0.pdf

Optica INAOE. (11 de Noviembre de 2020). Obtenido de https://www-optica.inaoep.mx/~jjbaezr/INAOECursos/colorimetria/pdefes/luz.pdf

Organización Mundial de la Salud. (1 de Diciembre de 2020). Obtenido de https://www.who.int/es/news-room/fact-sheets/detail/congenital-anomalies

Roberto CCU. (1 de Octubre de 2017). *El androide libre*. Obtenido de https://elandroidelibre.elespanol.com/2017/10/como-funciona-la-carga-inalambrica-que-es.html

Robles Vallejos, A. V. (Septiembre de 2020). *Universidad Nacional de Chimborazo.* Obtenido de http://dspace.unach.edu.ec/handle/51000/7091

Ródenas, L. (31 de Enero de 2014). Obtenido de https://drive.google.com/file/d/1VYTpHTjFeS-T-3YmX9fxulMeKFG7v5FW/view

RS Components. (19 de Noviembre de 2020). Obtenido de https://es.rs-online.com/web/c/semiconductores/circuitos-integrados-de-comunicacion-y-modulos-inalambricos/modulos-rf/#:~:text=El%20m%C3%B3dulo%20de%20RF%20es,un%20par%20de%20codificadores%2Fdecodificadores

Sanchez, A. (3 de Marzo de 2020). *FisioOnline*. Recuperado el 4 de octubre de 2019, de https://www.fisioterapia-online.com/articulos/que-es-la-hidroterapia-y-que-nos-puede-aportar

Sanz, F. (16 de Enero de 2019). *SaludTerapia.* Obtenido de https://www.saludterapia.com/glosario/d/32-fisioterapia.html

Secretaría de Salud. (05 de Marzo de 2018). *Secretaría de Salud*. Obtenido de https://www.gob.mx/salud/articulos/dia-mundial-de-los-defectos-de-nacimiento-149788?idiom=es

Sociedad Nacional del Síndrome de Down. (16 de Octubre de 2020). Obtenido de https://www.ndss.org/wp-content/uploads/2017/11/NDSS-GENERAL-BROCHURE-Spanish.pdf

TopDoctors. (6 de Noviembre de 2020). *TopDoctors España.* Obtenido de https://www.topdoctors.es/diccionario-medico/hidroterapia#

Xukyo. (18 de Diciembre de 2018). *AranaCorp*. Obtenido de https://www.aranacorp.com/es/tu-arduino-se-comunica-con-el-modulo-hc-05/

Anexo

A. Lista de materiales

Cantidad	Concepto	Valor unitario M.N. [$]	Total [$]
1	Arduino Mega 2560 con cable USB	299.00	299.00
1	Arduino Nano	229.00	229.00
12	Transistores irf640N	38.00	456.00
1	Tira LED RGB 5050 5 m	368.00	368.00
4	Buzzer	35.00	140.00
4	Acelerómetro MPU-6050	69.00	276.00
4	Transistor BC547B	45.00	180.00
1	Antena Bluetooth HC-05	131.00	131.00
5	Antena radiofrecuencia nRF24l01	120.00	600.00
14	Borneras 1x2	5.00	70.00
1	Transistor LD117V3.3	15.00	15.00
1	Cargador de batería inalámbrico	279.00	279.00
4	Receptor de carga inalámbrica	169.00	676.00
4	Batería 3.7 V de litio	39.00	156.00
4	18650 Shield V3	295.00	1180.00
75	Pinhead	3.50	262.00
4	Interruptor 2 polos 1 tiro	15.00	60.00
1	Interruptor 1 polo 1 tiro	32.00	32.00

1	Fuente Driver 240 W	406.00	406.00
4	Placa baquelita 10x10 cm	24.00	96.00
1	Placa baquelita 15x20 cm	57.00	57.00
1	Cloruro férrico	47.00	47.00
2	Impresión de pistas	17.00	34.00
Total			6119.00

B. Código del módulo trasmisor

```
#include <SPI.h>  //Librería para comunicarsecon el RF24.
#include <nRF24L01.h>
#include <RF24.h> //Librerías para el RF24.
#include <MPU6050.h>  //Librería para el MPU6050.
const int pinCE = 7;  //PIN para activar o desactivar el módulo.
const int pinCSN = 8;  //PIN para indicar al módulo si la comunicación es un mensaje o un comando.
const int pinAna = A0;  //PIN utilizado para el convertidor analógico-digital.
const int buz = 3;  //Variable para identificar el pin del zumbador.
RF24 radio(pinCE, pinCSN); //Creación del objeto nRF24.
#define Nodo 1   //Número para identificar al esclavo.
//Direcciones para los módulos.
const uint64_t dRadio[] = {0x7878787878LL, 0xB3B4B5B6F1LL, 0xB3B4B5B6CDLL,
0xB3B4B5B6A3LL, 0xB3B4B5B60FLL, 0xB3B4B5B605LL};
const uint64_t PTXpipe = dRadio[ Nodo - 1 ];   //Obtención de la dirección del nodo.
int DatoE = 0;  //Variable para guardar el código a enviar.
byte DatoR;  //Variable para guardar el dato recibido.
bool Buz = true;  //Variable para el buzzer.
int ref = 0;  //Se inicia la variable "ref" con un dato fuera de los rangos establecidos en el programa.
int Pila = 0;
float Valor = 0;  //Variables para el nivel de batería.
MPU6050 mpu;  //Declaración del sensor.
int16_t ax, ay, az, gx, gy, gz; //Variables del sensor.
//*****************************************************************//
const int numLecturas = 15; //Número de muestras para promedio.
```

```
int lecturas[numLecturas];  //Lecturas de la entrada analógica.
int indice = 0; //El índice de la lectura actual.
int total = 0;  //Total.
float promedioX = 0;  //Promedio.
//*********************************************************************//
const int numLecturas2 = 15; //Número de muestras para promedio.
int lecturas2[numLecturas2];  //Lecturas de la entrada analógica.
int indice2 = 0; //El índice de la lectura actual.
int total2 = 0;  //Total.
float promedioY = 0;  //Promedio.
//*********************************************************************//
const int numLecturas3 = 15; //Número de muestras para promedio.
int lecturas3[numLecturas3];  //Lecturas de la entrada analógica.
int indice3 = 0; //El índice de la lectura actual.
int total3 = 0;  //Total.
float promedioZ = 0;  //Promedio.
//*********************************************************************//
void setup()
{
  Serial.begin(9600);//Se incia el puerto Serial.
  radio.begin();//Inicio del módulo RF.
  radio.setRetries(15, 15);//Configuración del número máximo de intentos
  radio.setPALevel(RF24_PA_MIN);  //Nivel de transmisión. Se usa el nivel mínimo para pruebas.
  radio.setDataRate(RF24_250KBPS);//Se define la velocidad de envío de datos.
  radio.setChannel(108);//se fija un canal de comunicación para disminuir la interferencia.
  radio.openReadingPipe(0,PTXpipe);//Canal de lectura.
  radio.startListening();//Se inicia la escucha de datos. Se incia al esclavo como Receptor.
  pinMode(buz,OUTPUT); //Declarando salida para el zumbador.
  for (int lecturaA = 0; lecturaA < numLecturas; lecturaA++)//Se inician todas las lecturas a la entrada en cero.
    lecturas[lecturaA] = 0;
  for (int lecturaB = 0; lecturaB < numLecturas2; lecturaB++)
    lecturas2[lecturaB] = 0;
  for (int lecturaC = 0; lecturaC < numLecturas3; lecturaC++)
    lecturas3[lecturaC] = 0;
  mpu.initialize(); //Se inicia al acelerómetro.
  if (!mpu.testConnection())//Se comprueba que hay comunicación con el acelerómetro.
  {
```

```
    tone(buz, 600, 250);
  }
}
void loop()
{
  if(radio.available()) //Comprobación de datos entrantes en el módulo Rf.
  {
    radio.read(&DatoR, 1);  //Guardando los datos recibidos.
    asignaBuzzer(DatoR); //Subrutina de encendido-apagado del zumbador.
  }
  else
  {
    Bateria();
    mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz); //Se obtienen variables del sensor.
    ax = -(ax / 1000);  //Mapear monitor adelante-atrás.
    ay = (ay / 1000); //Mapear monitor izquierda-derecha.
    az = (az / 1000); //Mapear monitor arriba-abajo.
  //************************** Filtro eje X **************************//
    total = total - lecturas[indice]; //Se resta la última lectura.
    lecturas[indice] = ax; //Lecturas del sensor.
    total = total + lecturas[indice]; //Se añade la lectura al total.
    indice = indice + 1;  //Se avanza a la próxima posición del array.
    //Cuando se alcanza el final del array.
    if (indice >= numLecturas) indice = 0; //Se vuelve al inicio.
    //Se calcula el promedio.
    promedioX = total / numLecturas;  //Se manda a la PC como un valor ASCII.
  //************************** Filtro eje Y **************************//
    total2 = total2 - lecturas2[indice2]; //Se resta la última lectura.
    lecturas2[indice2] = ay; //Lecturas del sensor.
    total2 = total2 + lecturas2[indice2]; //Se añade la lectura al total.
    indice2 = indice2 + 1;  //Se avanza a la próxima posición del array.
    // Cuando se alcanza el final del array.
    if (indice2 >= numLecturas2) indice2 = 0; //Se vuelve al inicio.
    //Se calcula el promedio.
    promedioY = total2 / numLecturas2;  //Se manda a la PC como un valor ASCII.
  //************************** Filtro eje Z **************************//
    total3 = total3 - lecturas3[indice3]; //Se resta la última lectura.
```

```
    lecturas3[indice3] = az; //Lecturas del sensor.
    total3 = total3 + lecturas3[indice3]; //Se añade la lectura al total.
    indice3 = indice3 + 1;  //Se avanza a la próxima posición del array.
    //Cuando se alcanza el final del array.
    if (indice3 >= numLecturas3) indice3 = 0; //Se vuelve al inicio.
    //Se calcula el promedio.
    promedioZ = total3 / numLecturas3;  //Se manda a la PC como un valor ASCII.
  //*****************************************************************************//
    int X = promedioX;//Los datos son mapeados.
    X = map(X,-17,14,0,180);
    int Y = promedioY;
    Y = map(Y,-16,15,0,180);
    int Z = promedioZ;
    Z = map(Z,-15,17,0,180);
  //***************************//
    asignaCodigo(X, Y, Z, Pila);  //Condiciones para la selección y envío de datos.
  }
}
void asignaBuzzer(int R)  //Subrutina de encendido-apagado del zumbador.
{
  if(R == 1)  //Condiciones para prender/apagar el buzzer.
  {
    Buz = true;
    return Buz;
  }
  else{
    if(R == 2)
    {
      Buz = false;
      return Buz;
    }
  }
}
void asignaCodigo(int x, int y, int z, int B)  //Condiciones para la selección y envío de datos.
{
  if(x == 133 && B == 1)  //1 X
  {
```

```
 if(ref!=11)
 {
  radio.stopListening();  //Se pasa al modo de transmisor.
  radio.openWritingPipe(PTXpipe); //Se abre el canal de escritura.
  DatoE = 1; //Se crea la variable con el dato del código correspondiente.
  Serial.println(DatoE);
  if(radio.write(&DatoE, sizeof(DatoE)))  //Envío de datos.
  {
   if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
   else noTone(buz);            //valor retornado en la subrutina del mismo.
  }
  else{
   Serial.println("mal");  //Condición utilizada para pruebas. Sólo visible si se está conectado al PC.
  }
  ref = 11; //Se mantiene el estado de 'ref' hasta el siguiente cambio.
  delay(25);
  radio.startListening(); //Se regresa al modo de receptor.
 }
}
else
 if(x == 133 && B == 2)  //1 X
 {
  if(ref!=11)
  {
   radio.stopListening();  //Se pasa al modo de transmisor.
   radio.openWritingPipe(PTXpipe); //Se abre el canal de escritura.
   DatoE = 2; //Se crea la variable con el dato del código correspondiente.
   Serial.println(DatoE);
   if(radio.write(&DatoE, sizeof(DatoE)))  //Envío de datos.
   {
    if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
    else noTone(buz);            //valor retornado en la subrutina del mismo.
   }
   else{
    Serial.println("mal");  //Condición utilizada para pruebas. Sólo visible si se está conectado al PC.
   }
   ref = 11; //Se mantiene el estado de 'ref' hasta el siguiente cambio.
```

```
    delay(25);
    radio.startListening(); //Se regresa al modo de receptor.
   }
  }
 else
  if(x == 133 && B == 3)  //1 X
  {
    if(ref!=11)
    {
      radio.stopListening();  //Se pasa al modo de transmisor.
      radio.openWritingPipe(PTXpipe); //Se abre el canal de escritura.
      DatoE = 3; //Se crea la variable con el dato del código correspondiente.
      Serial.println(DatoE);
      if(radio.write(&DatoE, sizeof(DatoE)))  //Envío de datos.
      {
        if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
        else noTone(buz);            //valor retornado en la subrutina del mismo.
      }
      else{
        Serial.println("mal");  //Condición utilizada para pruebas. Sólo visible si se está conectado al PC.
      }
      ref = 11; //Se mantiene el estado de 'ref' hasta el siguiente cambio.
      delay(25);
      radio.startListening(); //Se regresa al modo de receptor.
    }
  }
  else
    if(x == 133 && B == 4)  //1 X
    {
      if(ref!=11)
      {
        radio.stopListening();  //Se pasa al modo de transmisor.
        radio.openWritingPipe(PTXpipe); //Se abre el canal de escritura.
        DatoE = 4; //Se crea la variable con el dato del código correspondiente.
        Serial.println(DatoE);
        if(radio.write(&DatoE, sizeof(DatoE)))  //Envío de datos.
        {
```

```
    if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
    else noTone(buz);            //valor retornado en la subrutina del mismo.
   }
   else{
    Serial.println("mal");  //Condición utilizada para pruebas. Sólo visible si se está conectado al PC.
   }
   ref = 11; //Se mantiene el estado de 'ref' hasta el siguiente cambio.
   delay(25);
   radio.startListening(); //Se regresa al modo de receptor.
  }
}
else
 if(y == 133 && B == 1)  //2 Y
 {
  if(ref!=12)
  {
   radio.stopListening(); //Se pasa al modo de transmisor.
   radio.openWritingPipe(PTXpipe);  //Se abre el canal de escritura.
   DatoE = 5;  //Se crea la variable con el dato del código correspondiente.
   if(radio.write(&DatoE, sizeof(DatoE))) //Envío de datos.
   {
    if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
    else noTone(buz);            //valor retornado en la subrutina del mismo.
   }
   else{
    Serial.println("mal"); //Condición utilizada para pruebas. Sólo visible si se está conectado al PC.
   }
   ref = 12;  //Se mantiene el estado de 'ref' hasta el siguiente cambio.
   delay(25);
   radio.startListening();  //Se regresa al modo de receptor.
  }
 }
 else
  if(y == 133 && B == 2)  //2 Y
  {
   if(ref!=12)
   {
```

```
   radio.stopListening(); //Se pasa al modo de transmisor.
   radio.openWritingPipe(PTXpipe);  //Se abre el canal de escritura.
   DatoE = 6;  //Se crea la variable con el dato del código correspondiente.
   if(radio.write(&DatoE, sizeof(DatoE))) //Envío de datos.
   {
    if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
    else noTone(buz);            //valor retornado en la subrutina del mismo.
   }
   else{
    Serial.println("mal"); //Condición utilizada para pruebas. Sólo visible si se está conectado al PC.
   }
   ref = 12;  //Se mantiene el estado de 'ref' hasta el siguiente cambio.
   delay(25);
   radio.startListening();  //Se regresa al modo de receptor.
  }
 }
 else
  if(y == 133 && B == 3)  //2 Y
  {
   if(ref!=12)
   {
    radio.stopListening(); //Se pasa al modo de transmisor.
    radio.openWritingPipe(PTXpipe);  //Se abre el canal de escritura.
    DatoE = 7;  //Se crea la variable con el dato del código correspondiente.
    if(radio.write(&DatoE, sizeof(DatoE))) //Envío de datos.
    {
     if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
     else noTone(buz);            //valor retornado en la subrutina del mismo.
    }
    else{
     Serial.println("mal"); //Condición utilizada para pruebas. Sólo visible si se está conectado al PC.
    }
    ref = 12;  //Se mantiene el estado de 'ref' hasta el siguiente cambio.
    delay(25);
    radio.startListening();  //Se regresa al modo de receptor.
   }
  }
```

```
else
 if(y == 133 && B == 4)  //2 Y
 {
  if(ref!=12)
  {
   radio.stopListening(); //Se pasa al modo de transmisor.
   radio.openWritingPipe(PTXpipe);  //Se abre el canal de escritura.
   DatoE = 8;  //Se crea la variable con el dato del código correspondiente.
   if(radio.write(&DatoE, sizeof(DatoE))) //Envío de datos.
   {
    if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
    else noTone(buz);            //valor retornado en la subrutina del mismo.
   }
   else{
    Serial.println("mal"); //Condición utilizada para pruebas. Sólo visible si se está conectado al PC.
   }
   ref = 12;  //Se mantiene el estado de 'ref' hasta el siguiente cambio.
   delay(25);
   radio.startListening();  //Se regresa al modo de receptor.
  }
 }
 else
  if(z == 135 && B == 1)  //3 Z
  {
   if(ref!=13)
   {
    radio.stopListening();  //Se pasa al modo de transmisor.
    radio.openWritingPipe(PTXpipe); //Se abre el canal de escrituta.
    DatoE = 9; //Se crea la variable con el dato del código correspondiente.
    if(radio.write(&DatoE, sizeof(DatoE)))  //Envío de datos.
    {
     if(Buz) tone(buz, 900, 175);  //El zumbador se activará dependiendo del
     else noTone(buz);            //valor retornado en la subrutina del mismo.
    }
    else{
     Serial.println("mal");  //Condición utilizada para pruebas. Sólo visible si se está conectado al.
    }
```

```
   ref = 13; //Se mantiene el estado de 'ref' hasta el siguiente cambio.
   delay(25);
   radio.startListening(); //Se regresa al modo de receptor.
  }
 }
 else
  if(z == 135 && B == 2)  //3 Z
  {
   if(ref!=13)
   {
    radio.stopListening();  //Se pasa al modo de transmisor.
    radio.openWritingPipe(PTXpipe); //Se abre el canal de escrituta.
    DatoE = 10; //Se crea la variable con el dato del código correspondiente.
    if(radio.write(&DatoE, sizeof(DatoE)))  //Envío de datos.
    {
     if(Buz) tone(buz, 900, 175);  //El zumbador se activará dependiendo del
     else noTone(buz);             //valor retornado en la subrutina del mismo.
    }
    else{
     Serial.println("mal");  //Condición utilizada para pruebas. Sólo visible si se está conectado al.
    }
    ref = 13; //Se mantiene el estado de 'ref' hasta el siguiente cambio.
    delay(25);
    radio.startListening(); //Se regresa al modo de receptor.
   }
  }
  else
   if(z == 135 && B == 3)  //3 Z
   {
    if(ref!=13)
    {
     radio.stopListening();  //Se pasa al modo de transmisor.
     radio.openWritingPipe(PTXpipe); //Se abre el canal de escrituta.
     DatoE = 11; //Se crea la variable con el dato del código correspondiente.
     if(radio.write(&DatoE, sizeof(DatoE)))  //Envío de datos.
     {
      if(Buz) tone(buz, 900, 175);  //El zumbador se activará dependiendo del
```

```
    else noTone(buz);            //valor retornado en la subrutina del mismo.
   }
   else{
    Serial.println("mal");  //Condición utilizada para pruebas. Sólo visible si se está conectado al
   }
   ref = 13; //Se mantiene el estado de 'ref' hasta el siguiente cambio.
   delay(25);
   radio.startListening(); //Se regresa al modo de receptor.
  }
 }
 else
  if(z == 135 && B == 4)  //3 Z
  {
   if(ref!=13)
   {
    radio.stopListening();  //Se pasa al modo de transmisor.
    radio.openWritingPipe(PTXpipe); //Se abre el canal de escrituta.
    DatoE = 12; //Se crea la variable con el dato del código correspondiente.
    if(radio.write(&DatoE, sizeof(DatoE)))  //Envío de datos.
    {
     if(Buz) tone(buz, 900, 175);  //El zumbador se activará dependiendo del
     else noTone(buz);            //valor retornado en la subrutina del mismo.
    }
    else{
     Serial.println("mal");  //Condición utilizada para pruebas. Sólo visible si se está conectado.
    }
    ref = 13; //Se mantiene el estado de 'ref' hasta el siguiente cambio.
    delay(25);
    radio.startListening(); //Se regresa al modo de receptor.
   }
  }
  else
   if(x == 46 && B == 1) //4 x
   {
    if(ref!=14)
    {
     radio.stopListening();   //Se pasa al modo de transmisor.
```

```
   radio.openWritingPipe(PTXpipe);  //Se abre el canal de lectura.
   DatoE = 13;  //Se crea la variable con el dato del código correspondiente.
   if(radio.write(&DatoE, sizeof(DatoE))) //Envío de datos.
   {
    if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
    else noTone(buz);            //valor retornado en la subrutina del mismo.
   }
   else{
    Serial.println("mal"); //Condición utilizada para prubeas. Sólo visible si se está conectado.
   }
   ref = 14;  //Se mantiene el estado de 'ref' hasta el siguiente cambio.
   delay(25);
   radio.startListening();  //Se regresa al modo de receptor.
  }
 }
 else
  if(x == 46 && B == 2) //4 x
  {
   if(ref!=14)
   {
    radio.stopListening();   //Se pasa al modo de transmisor.
    radio.openWritingPipe(PTXpipe);  //Se abre el canal de lectura.
    DatoE = 14;  //Se crea la variable con el dato del código correspondiente.
    if(radio.write(&DatoE, sizeof(DatoE))) //Envío de datos.
    {
     if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
     else noTone(buz);            //valor retornado en la subrutina del mismo.
    }
    else{
     Serial.println("mal"); //Condición utilizada para prubeas. Sólo visible si se está…
    }
    ref = 14;  //Se mantiene el estado de 'ref' hasta el siguiente cambio.
    delay(25);
    radio.startListening();  //Se regresa al modo de receptor.
   }
  }
  else
```

```
if(x == 46 && B == 3) //4 x
{
 if(ref!=14)
 {
  radio.stopListening();   //Se pasa al modo de transmisor.
  radio.openWritingPipe(PTXpipe);  //Se abre el canal de lectura.
  DatoE = 15;  //Se crea la variable con el dato del código correspondiente.
  if(radio.write(&DatoE, sizeof(DatoE))) //Envío de datos.
  {
   if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
   else noTone(buz);            //valor retornado en la subrutina del mismo.
  }
  else{
   Serial.println("mal"); //Condición utilizada para prubeas. Sólo visible si se está…
  }
  ref = 14;  //Se mantiene el estado de 'ref' hasta el siguiente cambio.
  delay(25);
  radio.startListening();  //Se regresa al modo de receptor.
 }
}
else
 if(x == 46 && B == 4) //4 x
 {
  if(ref!=14)
  {
   radio.stopListening();   //Se pasa al modo de transmisor.
   radio.openWritingPipe(PTXpipe);  //Se abre el canal de lectura.
   DatoE = 16;  //Se crea la variable con el dato del código correspondiente.
   if(radio.write(&DatoE, sizeof(DatoE))) //Envío de datos.
   {
    if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
    else noTone(buz);            //valor retornado en la subrutina del mismo.
   }
   else{
    Serial.println("mal"); //Condición utilizada para prubeas. Sólo visible si se está...
   }
   ref = 14;  //Se mantiene el estado de 'ref' hasta el siguiente cambio.
```

```
   delay(25);
   radio.startListening();  //Se regresa al modo de receptor.
  }
}
else
 if(y == 46 && B == 1) //5 y
 {
  if(ref!=15)
  {
   radio.stopListening();  //Se pasa al modo de transmisor.
   radio.openWritingPipe(PTXpipe); //Se abre el canal de escritura.
   DatoE = 17; //Se crea la variable con el dato del código correspondiente.
   if(radio.write(&DatoE, sizeof(DatoE)))  //Envío de datos.
   {
    if(Buz) tone(buz, 900, 175);  //El zumbador se activará dependiendo del
    else noTone(buz);             //valor retornado en la subrutina del mismo.
   }
   else{
    Serial.println("mal");  //Condición utilizada para pruebas. Sólo visible si se está...
   }
   ref = 15; //Se mantiene el estado de 'ref' hasta el siguiente cambio.
   delay(25);
   radio.startListening(); //Se regresa al modo de receptor.
  }
 }
 else
  if(y == 46 && B == 2) //5 y
  {
   if(ref!=15)
   {
    radio.stopListening();  //Se pasa al modo de transmisor.
    radio.openWritingPipe(PTXpipe); //Se abre el canal de escritura.
    DatoE = 18; //Se crea la variable con el dato del código correspondiente.
    if(radio.write(&DatoE, sizeof(DatoE)))  //Envío de datos.
    {
     if(Buz) tone(buz, 900, 175);  //El zumbador se activará dependiendo del
     else noTone(buz);             //valor retornado en la subrutina del mismo.
```

```
    }
    else{
      Serial.println("mal");  //Condición utilizada para pruebas. Sólo visible si se está...
    }
    ref = 15; //Se mantiene el estado de 'ref' hasta el siguiente cambio.
    delay(25);
    radio.startListening(); //Se regresa al modo de receptor.
  }
}
else
  if(y == 46 && B == 3) //5 y
  {
    if(ref!=15)
    {
      radio.stopListening();  //Se pasa al modo de transmisor.
      radio.openWritingPipe(PTXpipe); //Se abre el canal de escritura.
      DatoE = 19; //Se crea la variable con el dato del código correspondiente.
      if(radio.write(&DatoE, sizeof(DatoE)))  //Envío de datos.
      {
        if(Buz) tone(buz, 900, 175);  //El zumbador se activará dependiendo del
        else noTone(buz);             //valor retornado en la subrutina del mismo.
      }
      else{
        Serial.println("mal");  //Condición utilizada para pruebas. Sólo visible si se está…
      }
      ref = 15; //Se mantiene el estado de 'ref' hasta el siguiente cambio.
      delay(25);
      radio.startListening(); //Se regresa al modo de receptor.
    }
  }
  else
    if(y == 46 && B == 4) //5 y
    {
      if(ref!=15)
      {
        radio.stopListening();  //Se pasa al modo de transmisor.
        radio.openWritingPipe(PTXpipe); //Se abre el canal de escritura.
```

```
   DatoE = 20; //Se crea la variable con el dato del código correspondiente.
   if(radio.write(&DatoE, sizeof(DatoE)))  //Envío de datos.
   {
    if(Buz) tone(buz, 900, 175);  //El zumbador se activará dependiendo del
    else noTone(buz);             //valor retornado en la subrutina del mismo.
   }
   else{
    Serial.println("mal");  //Condición utilizada para pruebas. Sólo visible si se está...
   }
   ref = 15; //Se mantiene el estado de 'ref' hasta el siguiente cambio.
   delay(25);
   radio.startListening(); //Se regresa al modo de receptor.
  }
 }
 else
  if(z == 45 && B == 1) //6 z
  {
   if(ref!=16)
   {
    radio.stopListening(); //Se pasa al modo de transmisor.
    radio.openWritingPipe(PTXpipe);  //se abre el canal de escritura.
    DatoE = 21;  //Se crea la variable con el dato del código correspondiente.
    if(radio.write(&DatoE, sizeof(DatoE))) //Envío de datos.
    {
     if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
     else noTone(buz);            //valor retornado en la subrutina del mismo.
    }
    else{
     Serial.println("mal"); //Condición utilizada para pruebas. Sólo visible si se…
    }
    ref = 16;  //Se mantiene el estado de 'ref' hasta el siguiente cambio.
    delay(25);
    radio.startListening();  //Se regresa al modo de receptor.
   }
  }
  else
   if(z == 45 && B == 2) //6 z
```

```
{
 if(ref!=16)
 {
  radio.stopListening(); //Se pasa al modo de transmisor.
  radio.openWritingPipe(PTXpipe);  //se abre el canal de escritura.
  DatoE = 22;  //Se crea la variable con el dato del código correspondiente.
  if(radio.write(&DatoE, sizeof(DatoE))) //Envío de datos.
  {
   if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
   else noTone(buz);            //valor retornado en la subrutina del mismo.
  }
  else{
    Serial.println("mal"); //Condición utilizada para pruebas. Sólo visible si se…
  }
  ref = 16;  //Se mantiene el estado de 'ref' hasta el siguiente cambio.
  delay(25);
  radio.startListening();  //Se regresa al modo de receptor.
 }
}
else
 if(z == 45 && B == 3) //6 z
 {
  if(ref!=16)
  {
   radio.stopListening(); //Se pasa al modo de transmisor.
   radio.openWritingPipe(PTXpipe);  //se abre el canal de escritura.
   DatoE = 23;  //Se crea la variable con el dato del código correspondiente.
   if(radio.write(&DatoE, sizeof(DatoE))) //Envío de datos.
   {
    if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
    else noTone(buz);            //valor retornado en la subrutina del mismo.
   }
   else{
    Serial.println("mal"); //Condición utilizada para pruebas. Sólo visible si se…
   }
   ref = 16;  //Se mantiene el estado de 'ref' hasta el siguiente cambio.
   delay(25);
```

```
                radio.startListening();  //Se regresa al modo de receptor.
               }
              }
              else
               if(z == 45 && B == 4) //6 z
               {
                if(ref!=16)
                {
                 radio.stopListening(); //Se pasa al modo de transmisor.
                 radio.openWritingPipe(PTXpipe);  //se abre el canal de escritura.
                 DatoE = 24;  //Se crea la variable con el dato del código correspondiente.
                 if(radio.write(&DatoE, sizeof(DatoE))) //Envío de datos.
                 {
                  if(Buz) tone(buz, 900, 175); //El zumbador se activará dependiendo del
                  else noTone(buz);            //valor retornado en la subrutina del mismo.
                 }
                 else{
                  Serial.println("mal"); //Condición utilizada para pruebas. Sólo visible si se...
                 }
                 ref = 16;  //Se mantiene el estado de 'ref' hasta el siguiente cambio.
                 delay(25);
                 radio.startListening();  //Se regresa al modo de receptor.
                }
               }
}
void Bateria()
{
 float valor = analogRead(A0); //Convertidor analógico-digital.
 Valor = (valor * 1024)/3.7;
 //Condiciones de nivel de batería.
 if(Valor > 976.00)
 {
  Pila = 1;  //Asignación de código para nivel de batería.
 }
 else{
  if(Valor <= 976.00 && Valor > 928.00)
  {
```

```
    Pila = 2;    //Asignación de código para nivel de batería.
   }
   else{
    if(Valor <= 928.00 && Valor > 880.00)
    {
     Pila = 3;  //Asignación de código para el nivel de batería.
    }
    else{
     if(Valor <= 880)
     {
      Pila = 4;  //Asignación de código para el nivel de batería.
     }
    }
   }
  }
}
```

C. Código del módulo receptor

```
#include <SPI.h>  //Librería para comunicarse con el nRF24.
#include <nRF24L01.h>
#include <RF24.h> //Librerías para el nRF24.
const int pinCE = 48; //PIN para activar o desactivar al módulo.
const int pinCSN = 49; //PIN para indicar al módulo si la comunicación es un mensaje o un comando.
const int Led = 47; //Led que será usado para verificar el envío de datos.
RF24 radio(pinCE, pinCSN); //Creación del objeto nRF24.
const uint64_t dRadio[] = {0x7878787878LL, 0xB3B4B5B6F1LL, 0xB3B4B5B6CDLL,
0xB3B4B5B6A3LL, 0xB3B4B5B60FLL, 0xB3B4B5B605LL };  //Direcciones para los módulos.
int DatoR = 0;  //Variable para almacenar la información recibida de los esclavos.
int a, b, c, d; //Variables para identificar los niveles de batería en cada esclavo.
char Dato;  //Variable para almacenar la información presente en el puerto serial.

void setup()
{
 Serial.begin(9600); //Se incia el puerto serial.
 radio.begin();  //Inicio del módulo RF.
 radio.setRetries(15, 15); //Configuración del número máximo de reintentos.
 radio.setPALevel(RF24_PA_MIN);  //Nivel de transmisión. Se usa el nivel mínimo para pruebas.
 radio.setDataRate(RF24_250KBPS);  //Se define la velocidad de envío de datos.
```

```
  radio.setChannel(108);  //Se fija un canal de comunicación para disminuir la interferencia.
  radio.openReadingPipe(0, dRadio[0]);
  radio.openReadingPipe(1, dRadio[1]);
  radio.openReadingPipe(2, dRadio[2]);
  radio.openReadingPipe(3, dRadio[3]);  //Canales de lectura.
  radio.startListening();//Se incia al maestro como Receptor.
//Salidas PMW para activar las tiras led.
  pinMode(2, OUTPUT); // 1X
  pinMode(3, OUTPUT); // 1Y
  pinMode(4, OUTPUT); // 1Z
  pinMode(5, OUTPUT); // 2X
  pinMode(6, OUTPUT); // 2Y
  pinMode(7, OUTPUT); // 2Z
  pinMode(8, OUTPUT); // 3X
  pinMode(9, OUTPUT); // 3Y
  pinMode(10, OUTPUT); // 3Z
  pinMode(11, OUTPUT); // 4X
  pinMode(12, OUTPUT); // 4Y
  pinMode(13, OUTPUT); // 4Z
  pinMode(Led, OUTPUT); //Led de prueba.
}

void loop()
{
  if(radio.available()) //Verificando si hay recepción de datos.
  {
    while(radio.available())
    {
      radio.read(&DatoR, sizeof(DatoR));  //Lee el dato recibido y lo guarda en DatoR.
      Serial.println(DatoR);
      asignaEstados(DatoR);  //Subrutina de encendido de las tiras led.
    }
  }
  else{
    if (Serial.available()>0) //Si hay datos disponibles en el puerto serie bluetooth, los lee y los guarda
    {                         //en la variable 'dato' para poder usarlo según sea el caso.
      Dato = Serial.read(); //Guardando la información del puerto serial en la variable 'Dato'.
```

```
      asignaEstados2();  //Subrutina para encendido-apagado del zumbador presente en los esclavos.
      Bateria();  //Subrutina para envíar a la app el nivel de batería que tienen los esclavos.
      delay(50);
    }
  }
}
void asignaEstados(int codigo)
{
  switch(codigo)  //Selección e impresión de colores.
  {
    //Esclavo #1
    case 1:  //Rojo
    analogWrite(3,0);
    analogWrite(4,0);
    analogWrite(2,255);
    a = 11;
    break;
    case 2:  //Rojo
    analogWrite(3,0);
    analogWrite(4,0);
    analogWrite(2,255);
    a = 17;
    break;
    case 3:  //Rojo
    analogWrite(3,0);
    analogWrite(4,0);
    analogWrite(2,255);
    a = 15;
    break;
    case 4:  //Rojo
    analogWrite(3,0);
    analogWrite(4,0);
    analogWrite(2,255);
    a = 12;
    break;
    case 5:  //Verde
    analogWrite(2,0);
```

```
analogWrite(4,0);
analogWrite(3,255);
a = 11;
break;
case 6:  //Verde
analogWrite(2,0);
analogWrite(4,0);
analogWrite(3,255);
a = 17;
break;
case 7:  //Verde
analogWrite(2,0);
analogWrite(4,0);
analogWrite(3,255);
a = 15;
break;
case 8:  //Verde
analogWrite(2,0);
analogWrite(4,0);
analogWrite(3,255);
a = 12;
break;
case 9:  //Azul
analogWrite(2,0);
analogWrite(3,0);
analogWrite(4,255);
a = 11;
break;
case 10:  //Azul
analogWrite(2,0);
analogWrite(3,0);
analogWrite(4,255);
a = 17;
break;
case 11:  //Azul
analogWrite(2,0);
analogWrite(3,0);
```

```
analogWrite(4,255);
a = 15;
break;
case 12:  //Azul
analogWrite(2,0);
analogWrite(3,0);
analogWrite(4,255);
a = 12;
break;
case 13:  //Cian
analogWrite(2,0);
analogWrite(3,255);
analogWrite(4,255);
a = 11;
break;
case 14:  //Cian
analogWrite(2,0);
analogWrite(3,255);
analogWrite(4,255);
a = 17;
break;
case 15:  //Cian
analogWrite(2,0);
analogWrite(3,255);
analogWrite(4,255);
a = 15;
break;
case 16:  //Cian
analogWrite(2,0);
analogWrite(3,255);
analogWrite(4,255);
a = 12;
break;
case 17:  //Magenta
analogWrite(3,0);
analogWrite(2,255);
analogWrite(4,255);
```

```
a = 11;
break;
case 18:  //Magenta
analogWrite(3,0);
analogWrite(2,255);
analogWrite(4,255);
a = 17;
break;
case 19:  //Magenta
analogWrite(3,0);
analogWrite(2,255);
analogWrite(4,255);
a = 15;
break;
case 20:  //Magenta
analogWrite(3,0);
analogWrite(2,255);
analogWrite(4,255);
a = 12;
break;
case 21:  //Amarillo
analogWrite(4,0);
analogWrite(2,255);
analogWrite(3,255);
a = 11;
break;
case 22:  //Amarillo
analogWrite(4,0);
analogWrite(2,255);
analogWrite(3,255);
a = 17;
break;
case 23:  //Amarillo
analogWrite(4,0);
analogWrite(2,255);
analogWrite(3,255);
a = 15;
```

```
break;
case 24:  //Amarillo
analogWrite(4,0);
analogWrite(2,255);
analogWrite(3,255);
a = 12;
break;
//Esclavo #2
case 25:
analogWrite(6,0);
analogWrite(7,0);
analogWrite(5,255);
b = 11;
break;
case 26:
analogWrite(6,0);
analogWrite(7,0);
analogWrite(5,255);
b = 17;
break;
case 27:
analogWrite(6,0);
analogWrite(7,0);
analogWrite(5,255);
b = 15;
break;
case 28:
analogWrite(6,0);
analogWrite(7,0);
analogWrite(5,255);
b = 12;
break;
case 29:
analogWrite(5,0);
analogWrite(7,0);
analogWrite(6,255);
b = 11;
```

```
break;
case 30:
analogWrite(5,0);
analogWrite(7,0);
analogWrite(6,255);
b = 17;
break;
case 31:
analogWrite(5,0);
analogWrite(7,0);
analogWrite(6,255);
b = 15;
break;
case 32:
analogWrite(5,0);
analogWrite(7,0);
analogWrite(6,255);
b = 12;
break;
case 33:
analogWrite(5,0);
analogWrite(6,0);
analogWrite(7,255);
b = 11;
break;
case 34:
analogWrite(5,0);
analogWrite(6,0);
analogWrite(7,255);
b = 17;
break;
case 35:
analogWrite(5,0);
analogWrite(6,0);
analogWrite(7,255);
b = 15;
break;
```

```
case 36:
analogWrite(5,0);
analogWrite(6,0);
analogWrite(7,255);
b = 12;
break;
case 37:
analogWrite(5,0);
analogWrite(6,255);
analogWrite(7,255);
b = 11;
break;
case 38:
analogWrite(5,0);
analogWrite(6,255);
analogWrite(7,255);
b = 17;
break;
case 39:
analogWrite(5,0);
analogWrite(6,255);
analogWrite(7,255);
b = 15;
break;
case 40:
analogWrite(5,0);
analogWrite(6,255);
analogWrite(7,255);
b = 12;
break;
case 41:
analogWrite(6,0);
analogWrite(5,255);
analogWrite(7,255);
b = 11;
break;
case 42:
```

```
analogWrite(6,0);
analogWrite(5,255);
analogWrite(7,255);
b = 17;
break;
case 43:
analogWrite(6,0);
analogWrite(5,255);
analogWrite(7,255);
b = 15;
break;
case 44:
analogWrite(6,0);
analogWrite(5,255);
analogWrite(7,255);
b = 12;
break;
case 45:
analogWrite(7,0);
analogWrite(5,255);
analogWrite(6,255);
b = 11;
break;
case 46:
analogWrite(7,0);
analogWrite(5,255);
analogWrite(6,255);
b = 17;
break;
case 47:
analogWrite(7,0);
analogWrite(5,255);
analogWrite(6,255);
b = 15;
break;
case 48:
analogWrite(7,0);
```

```
analogWrite(5,255);
analogWrite(6,255);
b = 12;
break;
//Esclavo #3
case 49:
analogWrite(9,0);
analogWrite(10,0);
analogWrite(8,255);
c = 11;
break;
case 50:
analogWrite(9,0);
analogWrite(10,0);
analogWrite(8,255);
c = 17;
break;
case 51:
analogWrite(9,0);
analogWrite(10,0);
analogWrite(8,255);
c = 15;
break;
case 52:
analogWrite(9,0);
analogWrite(10,0);
analogWrite(8,255);
c = 12;
break;
case 53:
analogWrite(8,0);
analogWrite(10,0);
analogWrite(9,255);
c = 11;
break;
case 54:
analogWrite(8,0);
```

```
analogWrite(10,0);
analogWrite(9,255);
c = 17;
break;
case 55:
analogWrite(8,0);
analogWrite(10,0);
analogWrite(9,255);
c = 15;
break;
case 56:
analogWrite(8,0);
analogWrite(10,0);
analogWrite(9,255);
c = 12;
break;
case 57:
analogWrite(8,0);
analogWrite(9,0);
analogWrite(10,255);
c = 11;
break;
case 58:
analogWrite(8,0);
analogWrite(9,0);
analogWrite(10,255);
c = 17;
break;
case 59:
analogWrite(8,0);
analogWrite(9,0);
analogWrite(10,255);
c = 15;
break;
case 60:
analogWrite(8,0);
analogWrite(9,0);
```

```
analogWrite(10,255);
c = 12;
break;
case 61:
analogWrite(8,0);
analogWrite(9,255);
analogWrite(10,255);
c = 11;
break;
case 62:
analogWrite(8,0);
analogWrite(9,255);
analogWrite(10,255);
c = 17;
break;
case 63:
analogWrite(8,0);
analogWrite(9,255);
analogWrite(10,255);
c = 15;
break;
case 64:
analogWrite(8,0);
analogWrite(9,255);
analogWrite(10,255);
c = 12;
break;
case 65:
analogWrite(9,0);
analogWrite(8,255);
analogWrite(10,255);
c = 11;
break;
case 66:
analogWrite(9,0);
analogWrite(8,255);
analogWrite(10,255);
```

```
c = 17;
break;
case 67:
analogWrite(9,0);
analogWrite(8,255);
analogWrite(10,255);
c = 15;
break;
case 68:
analogWrite(9,0);
analogWrite(8,255);
analogWrite(10,255);
c = 12;
break;
case 69:
analogWrite(10,0);
analogWrite(8,255);
analogWrite(9,255);
c = 11;
break;
case 70:
analogWrite(10,0);
analogWrite(8,255);
analogWrite(9,255);
c = 17;
break;
case 71:
analogWrite(10,0);
analogWrite(8,255);
analogWrite(9,255);
c = 15;
break;
case 72:
analogWrite(10,0);
analogWrite(8,255);
analogWrite(9,255);
c = 12;
```

```
break;
//Esclavo #4
case 73:
analogWrite(45,0);
analogWrite(46,0);
analogWrite(44,255);
d = 11;
break;
case 74:
analogWrite(45,0);
analogWrite(46,0);
analogWrite(44,255);
d = 17;
break;
case 75:
analogWrite(45,0);
analogWrite(46,0);
analogWrite(44,255);
d = 15;
break;
case 76:
analogWrite(45,0);
analogWrite(46,0);
analogWrite(44,255);
d = 12;
break;
case 77:
analogWrite(44,0);
analogWrite(46,0);
analogWrite(45,255);
d = 11;
break;
case 78:
analogWrite(44,0);
analogWrite(46,0);
analogWrite(45,255);
d = 17;
```

```
break;
case 79:
analogWrite(44,0);
analogWrite(46,0);
analogWrite(45,255);
d = 15;
break;
case 80:
analogWrite(44,0);
analogWrite(46,0);
analogWrite(45,255);
d = 12;
break;
case 81:
analogWrite(44,0);
analogWrite(45,0);
analogWrite(46,255);
d = 11;
break;
case 82:
analogWrite(44,0);
analogWrite(45,0);
analogWrite(46,255);
d = 17;
break;
case 83:
analogWrite(44,0);
analogWrite(45,0);
analogWrite(46,255);
d = 15;
break;
case 84:
analogWrite(44,0);
analogWrite(45,0);
analogWrite(46,255);
d = 12;
break;
```

```
case 85:
analogWrite(44,0);
analogWrite(45,255);
analogWrite(46,255);
d = 11;
break;
case 86:
analogWrite(44,0);
analogWrite(45,255);
analogWrite(46,255);
d = 17;
break;
case 87:
analogWrite(44,0);
analogWrite(45,255);
analogWrite(46,255);
d = 15;
break;
case 88:
analogWrite(44,0);
analogWrite(45,255);
analogWrite(46,255);
d = 12;
break;
case 89:
analogWrite(45,0);
analogWrite(44,255);
analogWrite(46,255);
d = 11;
break;
case 90:
analogWrite(45,0);
analogWrite(44,255);
analogWrite(46,255);
d = 17;
break;
case 91:
```

```
    analogWrite(45,0);
    analogWrite(44,255);
    analogWrite(46,255);
    d = 15;
    break;
    case 92:
    analogWrite(45,0);
    analogWrite(44,255);
    analogWrite(46,255);
    d = 12;
    break;
    case 93:
    analogWrite(46,0);
    analogWrite(44,255);
    analogWrite(45,255);
    d = 11;
    break;
    case 94:
    analogWrite(46,0);
    analogWrite(44,255);
    analogWrite(45,255);
    d = 17;
    break;
    case 95:
    analogWrite(46,0);
    analogWrite(44,255);
    analogWrite(45,255);
    d = 15;
    break;
    case 96:
    analogWrite(46,0);
    analogWrite(44,255);
    analogWrite(45,255);
    d = 12;
    break;
  }
}
```

```
void asignaEstados2()
{
 if(Dato == '1')  //Condiciones para prender o apagar los buzzer.
 {
  byte val = 1;  //Dato a envíar.
  radio.stopListening();  //Maestro pasa a modo de transmisor.
  radio.openWritingPipe(dRadio[0]); //Dirección para el módulo transmisor #1.
  if(radio.write(&val, 1))  //Verificación del envío del dato.
  {
   digitalWrite(Led, HIGH);
   delay(500);
   digitalWrite(Led, LOW); //Condición para comprobar el envío del dato.
  }
  radio.startListening(); //Maestro regresa a modo de receptor.
 }
 else
  if(Dato == '2')
  {
   byte val = 2;  //Dato a envíar.
   radio.stopListening();  //Maestro pasa a modo de transmisor.
   radio.openWritingPipe(dRadio[0]); //Dirección para el módulo transmisor #1.
   if(radio.write(&val, 1))  //Verificación del envío del dato.
   {
    digitalWrite(Led, HIGH);
    delay(500);
    digitalWrite(Led, LOW); //Condición para comprobar el envío del dato.
   }
   radio.startListening();
  }
  else
   if(Dato == '3')
   {
    byte val = 1;  //Dato a envíar.
    radio.stopListening();  //Maestro pasa a modo de transmisor.
    radio.openWritingPipe(dRadio[1]); //Dirección para el módulo transmisor #2.
    if(radio.write(&val, 1))  //Verificación del envío del dato.
    {
```

```
    digitalWrite(Led, HIGH);
    delay(500);
    digitalWrite(Led, LOW); //Condición para comprobar el envío del dato.
   }
  radio.startListening(); //Maestro regresa a modo de receptor.
 }
 else
  if(Dato == '4')
  {
    byte val = 2;  //Dato a envíar.
    radio.stopListening();  //Maestro pasa a modo de transmisor.
    radio.openWritingPipe(dRadio[1]); //Dirección para el módulo transmisor #2.
    if(radio.write(&val, 1))  //Verificación del envío del dato.
    {
      digitalWrite(Led, HIGH);
      delay(500);
      digitalWrite(Led, LOW); //Condición para comprobar el envío del dato.
    }
    radio.startListening(); //Maestro regresa a modo de receptor.
  }
  else
    if(Dato == '5')
    {
      byte val = 1; //Dato a envíar.
      radio.stopListening(); //Maestro pasa a modo de transmisor.
      radio.openWritingPipe(dRadio[2]);  //Dirección para el módulo transmisor #3.
      if(radio.write(&val, 1)) //Verificación del envío del dato.
      {
        digitalWrite(Led, HIGH);
        delay(500);
        digitalWrite(Led, LOW); //Condición para comprobar el envío del dato.
      }
      radio.startListening();  //Maestro regresa a modo de receptor.
    }
    else
      if(Dato == '6')
      {
```

```
  byte val = 2;  //Dato a envíar.
  radio.stopListening();  //Maestro pasa a modo de transmisor.
  radio.openWritingPipe(dRadio[2]); //Dirección para el módulo transmisor #3.
  if(radio.write(&val, 1))  //Verificación del envío del dato.
  {
   digitalWrite(Led, HIGH);
   delay(500);
   digitalWrite(Led, LOW); //Condición para comprobar el envío del dato.
  }
  radio.startListening(); //Maestro regrea a modo de receptor.
 }
 else
  if(Dato == '7')
  {
   byte val = 1; //Dato a envíar.
   radio.stopListening(); //Maestro pasa a modo de transmisor.
   radio.openWritingPipe(dRadio[3]);  //Dirección para el módulo transmisor #4.
   if(radio.write(&val, 1)) //Verificación del envío del dato.
   {
    digitalWrite(Led, HIGH);
    delay(500);
    digitalWrite(Led, LOW); //Condición para comprobar el envío del dato.
   }
   radio.startListening();  //Maestro regresa a modo de receptor.
  }
  else
   if(Dato == '8')
   {
    byte val = 2;  //Dato a envíar.
    radio.stopListening();  //Maestro pasa a modo de transmisor.
    radio.openWritingPipe(dRadio[3]); //Dirección para el módulo transmisor #4.
    if(radio.write(&val, 1))  //Verificación del envío del dato.
    {
     digitalWrite(Led, HIGH);
     delay(500);
     digitalWrite(Led, LOW); //Condición para comprobar el envío del dato.
    }
```

```
                radio.startListening(); //Maestro regresa a modo de receptor.
               }
}
void Bateria()
{
  if (Dato == 'a')  //Condición para envíar el nivel de batería del esclavo #1.
  {
    Serial.print(a);
    Serial.print("\n");
  }
  else{
    if(Dato == 'b') //Condición para envíar el nivel de batería del esclavo #2.
    {
      Serial.print(b);
      Serial.print("\n");
    }
    else{
      if(Dato == 'c') //Condición para envíar el nivel de batería del esclavo #3.
      {
        Serial.print(c);
        Serial.print("\n");
      }
      else{
        if(Dato == 'd') //Condición para envíar el nivel de batería del esclavo #4.
        {
          Serial.print(d);
          Serial.print("\n");
        }
      }
    }
  }
}
```