UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE TELECOMUNICACIONES
DEPARTAMENTO DE SEÑALES Y SISTEMAS

ACTUALIZACIÓN DE UN PROTOTIPO DE ADQUISICIÓN DE DATOS DE MAGNITUDES FÍSICAS EN LOS NODOS DE REDIUC BASADO EN TECNOLOGÍA ARDUINO

ENMANUEL ENRIQUE RODRÍGUEZ PAZ
LUIS GUSTAVO PUENTES COLTELLACCI

Bárbula, 20 de Noviembre del 2015

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE TELECOMUNICACIONES
DEPARTAMENTO DE SEÑALES Y SISTEMAS

ACTUALIZACIÓN DE UN PROTOTIPO DE ADQUISICIÓN DE DATOS DE MAGNITUDES FÍSICAS EN LOS NODOS DE REDIUC BASADO EN TECNOLOGÍA ARDUINO

TRABAJO ESPECIAL DE GRADO PRESENTADO ANTE LA ILUSTRE UNIVERSIDAD DE CARABOBO PARA OPTAR AL TÍTULO DE INGENIERO DE TELECOMUNICACIONES

ENMANUEL ENRIQUE RODRÍGUEZ PAZ
LUIS GUSTAVO PUENTES COLTELLACCI

Bárbula, 20 de Noviembre del 2015

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE TELECOMUNICACIONES
DEPARTAMENTO DE SEÑALES Y SISTEMAS

AVAL DEL TUTOR

Quien suscribe Ing. Bill Torres, titular de la cédula de identidad 13.548.024, en mi carácter de TUTOR del Trabajo Especial de Grado titulado:

Actualización de un Prototipo de Adquisición de Datos de Magnitudes Físicas en los Nodos de REDIUC basado en tecnología Arduino

Y presentado por los bachilleres ENMANUEL ENRIQUE RODRÍGUEZ PAZ, cédula de identidad 21.098.240, LUIS GUSTAVO PUENTES COLTELLACCI, cédula de identidad 22.211.351, para optar al Título de Ingeniero de Telecomunicaciones, hago constar que dicho trabajo reúne los requisitos y méritos suficientes para ser sometido a la presentación pública y evaluación por parte del jurado examinador que se le designe

Firma
Prof. ING. BILL TORRES
CI: 13.548.024
TUTOR

Bárbula, 20 de Noviembre del 2015

UNIVERSIDAD DE CARABOBO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA DE TELECOMUNICACIONES
DEPARTAMENTO DE SEÑALES Y SISTEMAS

CERTIFICADO DE APROBACIÓN

Los abajo firmantes miembros del jurado asignado para evaluar el trabajo especial de grado titulado «ACTUALIZACIÓN DE UN PROTOTIPO DE ADQUISICIÓN DE DATOS DE MAGNITUDES FÍSICAS EN LOS NODOS DE REDIUC BASADO EN TECNOLOGÍA ARDUINO», realizado por los bachilleres ENMANUEL ENRIQUE RODRÍGUEZ PAZ, cédula de identidad 21.098.240, LUIS GUSTAVO PUENTES COLTELLACCI, cédula de identidad 22.211.351, hacemos constar que hemos revisado y aprobado dicho trabajo.

Firma
Prof. ING. BILL TORRES
TUTOR

Firma
Prof. NOMBRE DEL JURADO 1
JURADO

Firma
Prof. NOMBRE DEL JURADO 2
JURADO

Bárbula, 20 de Noviembre del 2015

Dedicatoria

A mis padres,
Aura Paz y Jorge Rodríguez,
mi apoyo incondicional

ENMANUEL ENRIQUE RODRÍGUEZ PAZ

A mis padres,
a mi hermano,
a mis amigos más allegados

LUIS GUSTAVO PUENTES COLTELLACCI

Agradecimientos

A nuestros padres, por el apoyo que nos brindaron a lo largo de nuestra carrera; al personal docente, administrativo y obrero de la Universidad de Carabobo, especialmente a los profesores de la Escuela de Telecomunicaciones por su entrega con la universidad; a nuestros amigos y compañeros de clases por compartir esta jornada con nosotros, y a todas y cada una de aquellas personas que de alguna manera contribuyeron en la consecución de este gran logro.

Índice general

Índice de figuras

Acrónimos

DIMETEL	**Di**rección de **M**edios **E**lectrónicos y **Tel**emática de la Universidad de Carabobo
IEEE	**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers
IP	**I**nternet **P**rotocol
LLC	**L**ogical **L**ink **C**ontrol
MAC	**M**edia **A**ccess **C**ontrol
NMS	**N**etwork **M**anagement **S**tations
REDIUC	**R**ed **D**orsal **D**igital **I**ntegrada de la **U**niversidad de **C**arabobo
SNMP	**S**imple **N**etwork **M**anagement **P**rotocol
UC	**U**niversidad de **C**arabobo

ACTUALIZACIÓN DE UN PROTOTIPO DE ADQUISICIÓN DE DATOS DE MAGNITUDES FÍSICAS EN LOS NODOS DE REDIUC BASADO EN TECNOLOGÍA ARDUINO

por

ENMANUEL ENRIQUE RODRÍGUEZ PAZ y LUIS GUSTAVO PUENTES COLTELLACCI

Presentado en el Departamento de Señales y Sistemas
de la Escuela de Ingeniería en Telecomunicaciones
el 20 de Noviembre del 2015 para optar al Título de
Ingeniero de Telecomunicaciones

RESUMEN

Este trabajo fue desarrollado con miras a implementar en el nodo de comunicaciones de Bárbula de la Universidad de Carabobo un prototipo de monitoreo de variables de interés para los administradores de la red, capaz de incorporarse a la red ya existente de la universidad, REDIUC. El prototipo se desarrolló para que fuese capaz de detectar presencia de humo, de calor, niveles normales de presiones alta y baja de un aire acondicionado, detección de apertura/cierre de la puerta del nodo y obtener mediciones de temperatura y humedad, y con la intención de ser

replicable en otros nodos de comunicaciones de la universidad buscándose entonces una construcción sencilla y de costo asequible. El prototipo fue implementado haciendo uso de tecnología Arduino, realizando una propuesta de los sensores que se le habrían de incorporar y un esquema de conexión para estos. Se desarrolló de un código que permitiese comunicación mediante el protocolo SNMP de monitoreo, permitiendo al prototipo funcionar como un agente SNMP capaz de responder a solicitudes *get*, *set* y *walk*, y de enviar mensajes *trap* cuando detecta alguna anomalía en el nodo. Se hizo un montaje de los elementos el prototipo y este fue instalado en nodo de comunicaciones de Bárbula. El funcionamiento del prototipo se puso a prueba con varios programas en los sistemas operativos Windows para ordenadores y Android para teléfonos, denotando compatibilidad con la mayoría de estas herramientas a pesar de presentar una serie de limitaciones que se mencionan en el trabajo. El prototipo fue incorporado a la red universitaria y a los softwares de monitoreo que se utilizan en esta, dejándose a disposición de los administradores de la red junto con un CD que contiene los códigos y librerías que utiliza y un manual de implementación que resume las características del prototipo para simplificar su réplica, modificación, ampliación de capacidades, limitaciones y corrección de errores.

Palabras Claves: Arduino, SNMP

Tutor: ING. BILL TORRES
Profesor del Departamento de Señales y Sistemas
Escuela de Telecomunicaciones. Facultad de Ingeniería

Capítulo I

El Problema

1.1. Motivación

La Universidad de Carabobo que a manera de cubrir las expectativas de brindar una formación universitaria de calidad y una base sólida para el desarrollo de investigaciones ha implementado en su infraestructura la Red Dorsal Digital Integrada de la Universidad de Carabobo (REDIUC). A través de esta plataforma, se logra la interconexión de las distintas facultades y dependencias de la UC, así como también la conexión hacia la red mundial de redes (Internet) y la red académica nacional e internacional (Internet2, CLARA, RedIRIS, Géant, etc.). Además permite la transmisión de la radio y televisión universitaria por señal abierta y cable respectivamente, servicios de telefonía fija interna e interinstitucional (VoIP) y videoconferencia.

Actualmente la universidad cuenta con 18 nodos de telecomunicaciones operativos ubicados a lo largo de los municipios Valencia y Naguanagua del estado Carabobo. Cada uno de estos nodos está conformado por equipos costosos y sensibles a factores externos como lo son la temperatura y humedad, por lo que deben funcionar en un ambiente controlado y seguro para mantener su integridad física. Por otro lado, existe actualmente en la universidad una problemática de inseguridad que pone en riesgo a los equipos ubicados en los nodos, junto con el hecho de

que estos son equipos que deben tener un acceso restringido y se les debe prestar un debido mantenimiento. Un administrador de red debe ser capaz de monitorear estas condiciones de cada nodo en tiempo real y preferiblemente desde la comodidad de su computador personal o incluso desde su dispositivo móvil. La Universidad de Carabobo actualmente no cuenta con un sistema de monitoreo de las condiciones dentro de los nodos de telecomunicaciones y esto provoca entre otras cosas que no se puedan detectar anomalías dentro de los mismos.

Por los motivos presentados anteriormente León S. y Loaiza E. (2002) [1] implementaron un prototipo de adquisición de datos de magnitudes físicas en el nodo ubicado en Bárbula, con la intención de que se pudiese tener conocimiento remoto de las condiciones que presentaba la habitación de comunicaciones del nodo, y que con visión a futuro el prototipo se implementara en los demás nodos. Este prototipo consistía en un sistema de monitoreo mediante sensores que permitía la medición de temperatura, la detección de humo, la medición de la presión del gas de refrigeración en los aires acondicionados, la entrada y salida de una persona al nodo, entre otras cosas y mandarlas mediante una conexión Ethernet a la sede principal donde serían monitoreadas por un usuario autorizado. Actualmente este prototipo se encuentra fuera de servicio, por lo que se ha optado por desarrollar un nuevo prototipo de monitoreo capaz de cubrir con las funciones del original y otras nuevas valiéndose de los avances tecnológicos que han ocurrido en los últimos años, de una manera rentable y sencilla de implementar.

Se toma de ejemplo el caso de Castro A. (2013) [2], quien plantea un sistema de control de temperatura basado en Arduino por su fácil acceso y por poseer una infinidad de aplicaciones, y así como lo plantea Sánchez E. (2012) [3] con la plataforma Arduino se puede crear un sistema de monitoreo similar a los que se instalan en la actualidad pero a un costo mucho menor, dados todos los módulos sensores que se le pueden incorporar al Arduino. Por estos motivos se ha decidido optar por la plataforma Arduino de hardware libre como base para el desarrollo del prototipo con la finalidad de preservar el estado físico de los equipos de un nodo alargando así su vida útil, protegiendo las inversiones de la Universidad de Carabobo y asegurando el buen servicio que ofrece REDIUC.

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar un prototipo actualizado de adquisición de datos de magnitudes físicas en el nodo de REDIUC ubicado en la Facultad de Ingeniería basado en tecnología Arduino.

1.2.2. Objetivos Específicos

- Seleccionar los dispositivos necesarios para realizar la medición de las magnitudes planteadas.
- Diseñar un código para el procesamiento de los datos adquiridos basado en el uso de la tecnología Arduino.
- Implementar el prototipo en el nodo de REDIUC ubicado en la Facultad de Ingeniería de la Universidad de Carabobo.
- Desarrollar un manual de implementación del prototipo.

1.3. Alcances

Con el prototipo generado de este proyecto se logrará el monitoreo de datos de magnitudes físicas definidas mediante el protocolo de red SNMP y será implementado en el nodo Bárbula de REDIUC en la Facultad de Ingeniería. Las magnitudes físicas del nodo que podrán monitorearse son la temperatura, la humedad relativa, la apertura/cierre de la puerta, la presión alta/baja de un aire acondicionado y la detección de incendio mediante sensores de humo y calor, quedando posibilidad para la detección de nuevas variables o la implementación de nuevos elementos. Se desarrollará un manual de implementación que donde se especifiquen los elementos que conforman en el prototipo, su esquema de conexión final y el código

involucrado en su funcionamiento, con la finalidad de facilitar la fabricación e implementación de réplicas en los otros nodos de REDIUC, la detección de daños en el prototipo y la ampliación de sus capacidades.

Capítulo II

Marco conceptual

2.1. Sensores

Un sensor es un dispositivo que detecta o mide una cantidad física, generalmente produciendo una señal eléctrica con la finalidad de detectar o medir [4].

2.1.1. Tipos de Sensores

Los sensores se presentan en una gran variedad, pero según las características que presentan se pueden clasificar de la siguiente manera:

- Según el aporte de energía:
  Moduladores o pasivos: la energía de la señal de salida procede, en su mayor parte, de una fuente de energía auxiliar. La entrada sólo controla la salida.
  Generadores o activos: la energía de salida es suministrada por la entrada.

- Según la señal de salida:
  Analógicos: la salida varía, a nivel macroscópico, de forma continua. La información está en la amplitud, si bien se suelen incluir en este grupo los sensores con salida en el dominio temporal.

Digitales: la salida varía en forma de saltos o pasos discretos. No requieren conversión A/D y la transmisión de su salida es más fácil.

- Según el modo de funcionamiento:
  De deflexión: la magnitud producida produce algún efecto físico, que engendra algún efecto similar, pero opuesto, en alguna parte del instrumento, y que está relacionado con alguna variable útil.
  De comparación: se intenta mantener nula la deflexión mediante la aplicación de un efecto bien conocido, opuesto al generado por la magnitud a medir. Hay un detector del desequilibrio y un medio para restablecerlo.

- Según el tipo de relación entrada-salida: pueden ser de orden cero, de primer orden, de segundo orden o de orden superior. El orden está relacionado con el número de elementos almacenadores de energía independientes que incluye el sensor, y repercute en su exactitud y velocidad de respuesta. Esta clasificación es de gran importancia cuando el sensor forma parte de un sistema de control en lazo cerrado.

2.1.2. Características Estáticas de un Sensor

Son aquellas características que pueden medirse después de que todos los efectos transitorios se han estabilizado a un estado final o estable [5]. Se pueden resaltar las siguientes:

- Exactitud: define qué tan correctamente la salida del sensor representa el valor verdadero.

- Error: es la diferencia entre el valor verdadero de la cantidad siendo medida y el valor actual obtenido del sensor.

- Presición: es la capacidad de un instrumento de dar el mismo resultado en mediciones diferentes realizadas en las mismas condiciones.

- Resolución: indica el menor cambio incremental variable a medir que resultará en un incremento detectable en la señal de salida. La resolución se encuentra fuertemente limitada por cualquier ruido en la señal.
- Sensibilidad: es la tasa de cambio incremental en la salida del sensor a un cambio incremental en la variable a medir en la entrada.
- Selectividad: es la habilidad de un sensor de medir un único componente en la presencia de otros. Por ejemplo, un sensor de oxígeno que no muestre respuesta a otros gases como CO, CO2 y NO2 podría considerarse como selectivo.
- Límite de Detección: es la menor magnitud de la variable a medir que puede ser percibida por el sensor.
- Respuesta en el Tiempo: es el tiempo que le toma al sensor alcanzar un valor estable. Generalmente se expresa en el tiempo en el cual la salida alcanza un cierto porcentaje de su valor final (por ejemplo 95 %), en respuesta a un cambio brusco en la entrada.
- Rango Dinámico o Span: el rango de señales de entrada que resultarán en una salida significativa del sensor. Todos los sensores están diseñados para funcionar dentro de un rango específico. Las señales fuera de ese rango pueden ser ininteligibles, causar inaceptables grandes errores en la medición o incluso resultar en daños irreparables en el sensor.

2.2. Arduino

2.2.1. Definición

Es una plataforma electrónica de código abierto de software y hardware para el sencillo desarrollo de proyectos interactivos [6]. Entre los productos Arduino se pueden encontrar tarjetas, expansiones, kits y accesorios; las tarjetas son capaces de interactuar con su ambiente al ser capaces de recibir las mediciones de distintos sensores y controlar luces, motores y demás dispositivos.

2.2.2. Modelos

Las placas Arduino fueron creadas con la intención de permitir al diseñador realizar cualquier proyecto que se plantee, abarcando la mayoría o totalidad de funciones que este puede requerir. Es por esto que existe una variedad de placas Arduino, de manera de ofrecer al diseñador el producto que mejor se adapte a sus proyectos. Con esto en mente hoy día se pueden encontrar en el mercado los siguientes modelos de placas Arduino [6]:

- Arduino Uno: es una tarjeta electrónica basada en el ATmega328. Posee 14 pines de entradas/salidas digitales (de los cuales 6 pueden usarse como salidas PWM), 6 entradas analógicas, un resonador cerámico de 16 MHz, una conexión USB, un puerto para energizar, una cabecera ICSP y un botón de reinicio.

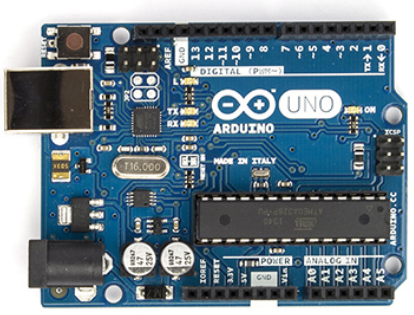


Figura 2.1: Arduino Uno

- Arduino Leonardo: es una tarjeta electrónica basada en el ATmega32u4. Posee 20 pines de entradas/salidas digitales (de las cuales 7 pueden usarse como salidas PWM y 12 como entradas analógicas), un oscilador de cristal de 16 MHz, una conexión micro USB, un puerto para energizar, una cabecera ICSP y un botón de reinicio.
- Arduino Due: es una tarjeta electrónica basada en el Atmel SAM3X8E ARM Cortex-M3. Es la primera tarjeta Arduino basada en un núcleo microcontrolador ARM de 32 bits. Posee 54 pines de entradas/salidas digitales (de las cuales 12 pueden usarse como salidas PWM), 12 entradas analógicas, 4 UARTs

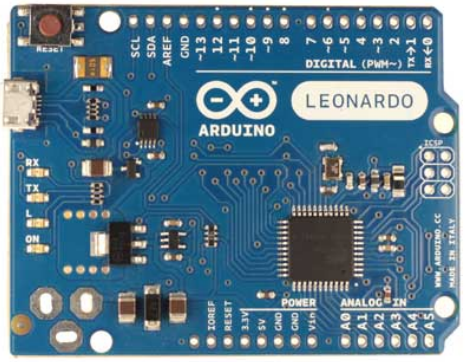


Figura 2.2: Arduino Leonardo

(puertos seriales), un reloj de 84 MHz, una conexión USB con capacidad de enlaces OTG (On The Go), 2 conversores digitales a analógicos, 2 TWI (Two Wire Interface, derivación de interfaz I²C), un puerto para energizar, una cabecera SPI, una cabecera JTAG, un botón de reinicio y un botón de borrado.

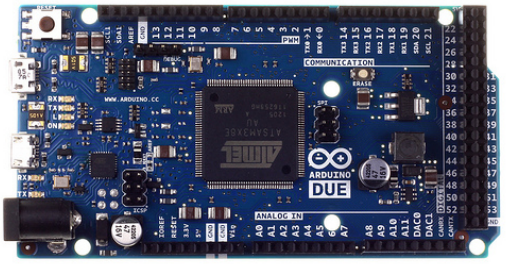


Figura 2.3: Arduino Due

- Arduino Yún: es una tarjeta electrónica basada en el ATmega32u4 y el Atheros AR9331. El procesador Atheros soporta una distribución Linux basada en OpenWrt llamado OpenWrt-Yun. La tarjeta posee soportes Ethernet y WiFi incorporados, un puerto USB-A, una ranura micro-SD, 20 pines de entradas/-salidas digitales (de los cuales 7 pueden usarse como salidas PWM y 12 como entradas analógicas). Un oscilador de cristal de 16 MHz, una conexión micro USB, una cabecera ICSP y 3 botones de reinicio.

Figura 2.4: Arduino Yún

- Arduino Micro: es una tarjeta electrónica basada en el ATmega32u4. Posee 20 pines de entradas/salidas digitales (de las cuales 7 pueden usarse como salidas PWM y 12 como entradas analógicas), un oscilador de cristal de 16 MHz, una conexión micro USB, una cabecera ICSP y un botón de reinicio.

Figura 2.5: Arduino Micro

- Arduino Esplora: es una tarjeta electrónica derivada del Arduino Leonardo. El Esplora difiere de todas las tarjetas Arduino precedentes en que provee un número de sensores incorporados para interacción listos para usarse. Está diseñado para personas que quieren empezar a trabajar con Arduino tener que aprender primero sobre electrónica. Posee salidas incorporadas para luces y sonidos y varios sensores de entrada, incluyendo una palanca, un control deslizante, un sensor de temperatura, un acelerómetro, un micrófono y un sensor de luz.
- Arduino Mega ADK: es una tarjeta electrónica basada en el ATmega2560. Posee una interface host USB para conectar con teléfonos basados en Android, basada en el MAX3421e IC. Posee 54 pines de entradas/salidas digitales (de

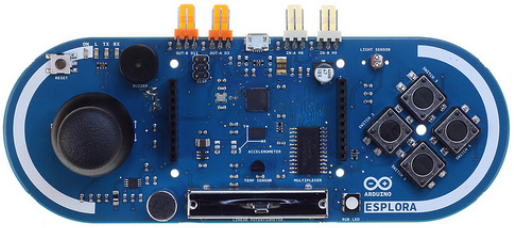


Figura 2.6: Arduino Esplora

las cuales 15 pueden usarse como salidas PWM), 16 entradas analógicas, 4 UARTs (puertos seriales), un oscilador de cristal de 16 MHz, una conexión USB, un puerto para energizar, una cabecera ICSP y un botón de reinicio.

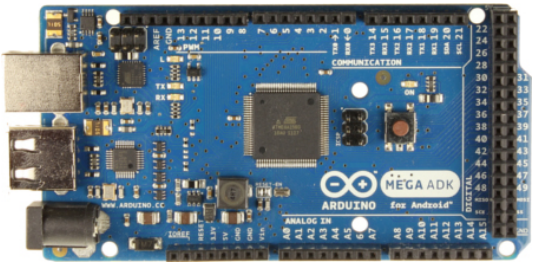


Figura 2.7: Arduino Mega ADK

- Arduino Ethernet: es una tarjeta electrónica basada en el ATmega328. Posee 14 pines de entradas/salidas digitales, 6 entradas analógicas, un oscilador de cristal de 16 MHz, una conexión RJ45, un puerto para energizar, una cabecera ICSP y un botón de reinicio. Los pines 10, 11, 12 y 13 están reservados para el módulo de interfaz Ethernet y no deben usarse de otra manera, lo que reduce el número de pines disponibles a 9, 4 de ellos disponibles para salidas PWM. El Arduino Ethernet difiere de otras tarjetas en que no posee un chip incorporado para conversión USB a serial, sino que posee una interfaz Ethernet Wiznet, que es la misma utilizada en los módulos Ethernet. Posee un lector de tarjetas micro SD incorporado, que puede utilizarse para almacenar archivos de servidor para una red.

- Arduino Mega 2560: es una tarjeta electrónica basada en el ATmega2560. Posee 54 pines de entradas/salidas digitales (de los cuales 15 pueden utilizarse como salidas PWM), 16 entradas analógicas, 4 UARTs (puertos seriales), un

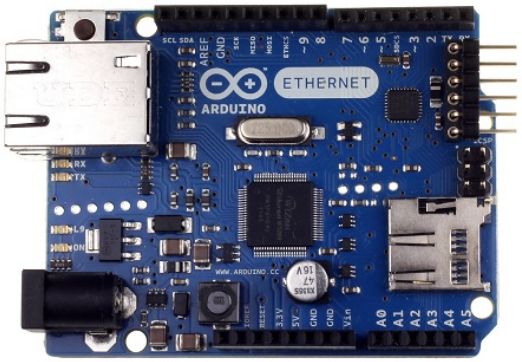


Figura 2.8: Arduino Ethernet

oscilador de cristal de 16 MHz, una conexión USB, un puerto para energizar, una cabecera ICSP y un botón de reinicio.

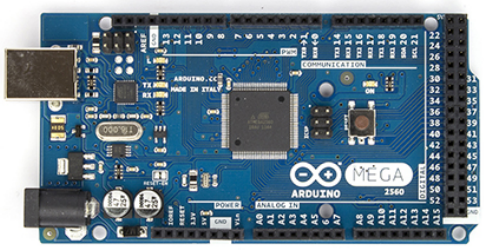


Figura 2.9: Arduino Mega 2560

- Arduino Robot: es el primer Arduino oficial sobre ruedas. El robot posee dos procesadores, uno en cada una de sus dos placas. La placa motor controla los motores y la placa control lee sensores y decide cómo operar. Cada una de las placas es una placa Arduino completamente programable, y ambas se basan en el ATmega32u4. Posee una velocidad de reloj de 16 MHz, 5 pines de entradas/salidas digitales (de los cuales 4 se pueden utilizar para entradas analógicas), un teclado de 5 botones, una corneta y un botón de reinicio.
- Arduino Mini: : es una pequeña tarjeta electrónica actualmente basada en el ATmega328, previsto para el uso en placas de prueba (como protoboards) o

Figura 2.10: Arduino Robot

cuando se busca optimizar el espacio. Posee 14 pines de entradas/salidas digitales (de los cuales 6 pueden usarse como salidas PWM), 8 entradas analógicas y un oscilador de cristal de 16 MHz. Puede programarse con un adaptador USB serial u otro adaptador serial USB o RS232 a TTL. La versión más actual posee un botón de reinicio incorporado.

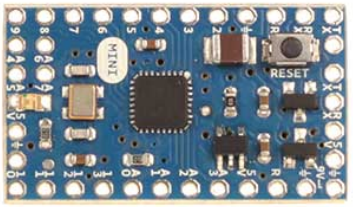


Figura 2.11: Arduino Mini

- Arduino Nano: es una tarjeta electrónica pequeña, completa y amigable con placas de prueba, basada en el ATmega328 (Arduino Nano 3.x) o en el ATmega168 (Arduino Nano 2.x). Posee 14 pines de entradas/salidas digitales (de los cuales 6 pueden usarse como salidas PWM), 8 salidas analógicas, un reloj de 16 MHz, una conexión USB, una cabecera ICSP y un botón de reinicio. Le falta únicamente el puerto de energización DC y funciona con un cable USB Mini-B en vez de una estándar.

Figura 2.12: Arduino Nano

- LilyPad Arduino: es una tarjeta electrónica diseñada para vestimentas y textiles electrónicos. Está basada en el ATmega168V o el ATmega328V. Posee 14 pines de entradas/salidas digitales (de las cuales 6 proveen salida PWM), 6 entradas analógicas, posee una velocidad de reloj de 8 MHz, un botón de reinicio y puede energizarse vía conexión USB o con una fuente de energía externa.

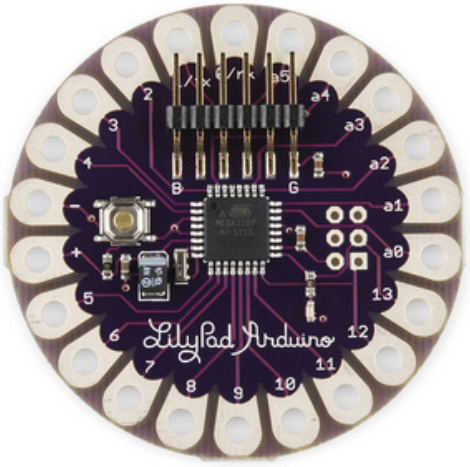


Figura 2.13: LilyPad Arduino

- LilyPad Arduino Simple: es una tarjeta electrónica diseñada para vestimentas y textiles electrónicos. Puede ser cocido a la tela y de manera similar se le pueden conectar fuentes de alimentación, sensores y actuadores con hilo conductor. Posee una velocidad de reloj de 8 MHz y a diferencia de la tarjeta

LilyPad Arduino original esta posee sólo 9 pines de entradas/salidas (de los cuales 5 pueden usarse como salidas PWM y 4 como entradas analógicas). Posee además un botón de reinicio, un conector JST y un circuito de carga incorporado para baterías de polímeros de litio. La tarjeta está basada en el ATmega328.

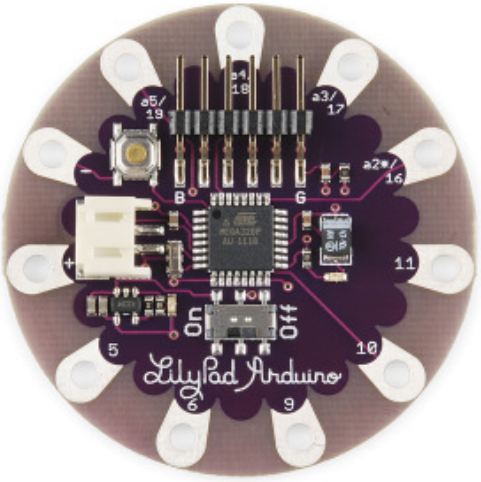


Figura 2.14: LilyPad Arduino Simple

- LilyPad Arduino SimpleSnap: es una tarjeta electrónica diseñada para vestimentas y textiles electrónicos. Es similar al LilyPad Arduino Simple, excepto que posee una batería de polímero de lito incorporada y en vez de agujeros posee broches conductivos. Al utilizarse broches de este tipo en un proyecto se puede fijar la tarjeta de manera segura y removerse para lavar la vestimenta o moverse a otra nueva. Posee una velocidad de reloj de 8 MHz, un botón de reinicio y posee sólo 9 pines de entradas/salidas digitales (de los cuales 5 pueden usarse como salidas PWM y 4 como entradas analógicas). Adicionalmente posee incorporado un circuito de carga para la batería.
- LilyPad Arduino USB: es una tarjeta electrónica basada en el ATmega32u4. Posee 9 pines de entradas/salidas digitales (de los cuales 4 pueden usarse como salidas PWM y 4 como entradas analógicas), un resonador de 8 MHz, una conexión micro USB, un conector JST para una batería de 3.7 voltios de LiPo y un botón de reinicio. Esta tarjeta difiere de las otras tarjetas LilyBoard

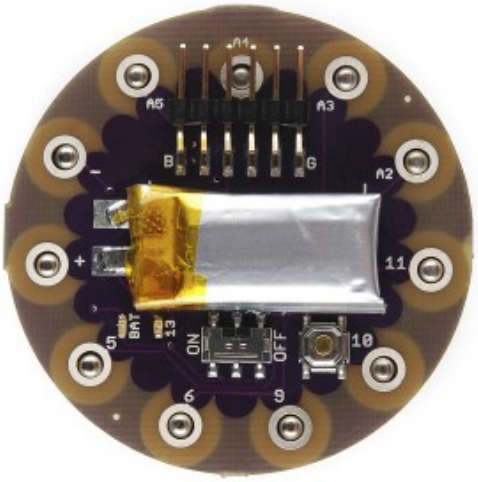


Figura 2.15: LilyPad Arduino SimpleSnap

en que el ATmega32u4 posee comunicación USB incorporada, eliminando la necesidad de un adaptador USB -serial separado y facilitando de esta manera la manera de conectarse a la computadora y programarse.

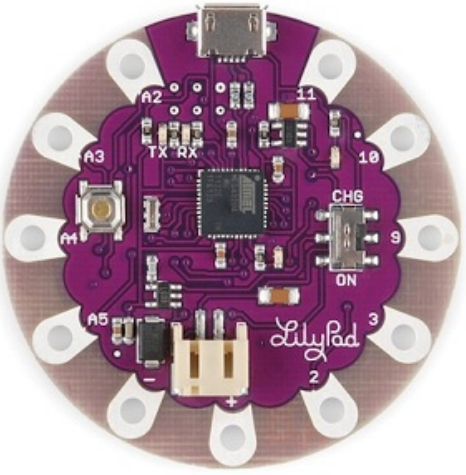


Figura 2.16: LilyPad Arduino USB

- Arduino Pro Mini: es una tarjeta electrónica basada en el ATmega328. Posee 14 pines de entradas/salidas digitales (de los cuales 6 pueden usarse como salidas PWM), 6 entradas analógicas, un resonador de 8 o 16 MHz (según la

versión), un botón de reinicio y agujeros para montar cabeceras de pines (no posee cabeceras previamente montadas). Está diseñado con la intención de usarse para instalaciones semipermanentes en objetos o exhibiciones.

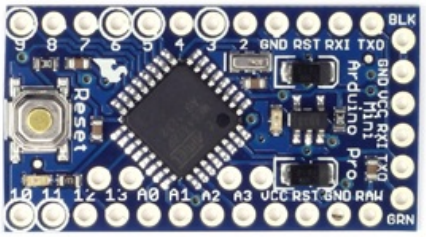


Figura 2.17: Arduino Pro Mini

- Arduino Fio: es una tarjeta microelectrónica basada en el ATmega328P. Funciona con un resonador incorporado de 8 MHz y posee 14 pines de entradas/salidas digitales (de los cuales 6 pueden usarse como salidas PWM), 8 entradas analógicas, un botón de reinicio y agujeros para el montaje de cabeceras de pines (no posee cabeceras previamente montadas). Posee conexiones para una batería de polímero de litio e incluye un circuito de carga a través de USB. Posee un enchufe XBee (para módulos de radio) en la base. Está diseñado con la intención de usarse en aplicaciones inalámbricas.

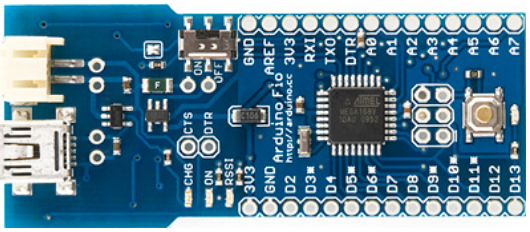


Figura 2.18: Arduino Fio

- Arduino Pro: es una tarjeta electrónica basada en el ATmega168 o el ATmega328. Viene en una versión de 3,3 voltios/8 MHz y una versión de 5 voltios/16 MHz. Posee 14 pines de entradas/salidas digitales (de los cuales 6 pueden usarse como salidas PWM), 6 entradas analógicas, un puerto de conexión de batería, un switch de encendido, un botón de reinicio y agujeros

para el montaje de un puerto de energización, una cabecera ICSP y cabeceras de pines. Esta tarjeta está diseñada con la intención con la intención de usarse para instalaciones semipermanentes en objetos o exhibiciones.

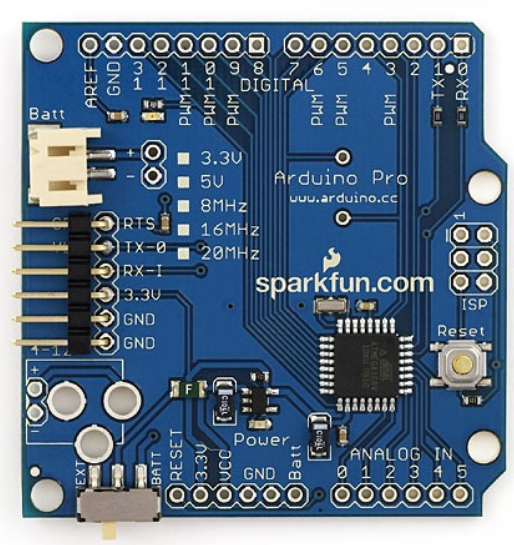


Figura 2.19: Arduino Pro

2.2.3. Lenguaje de Programación

La plataforma Arduino se programa mediante el uso de un lenguaje propio basado en el lenguaje de programación de alto nivel Processing. Al escribir un código en la IDE de Arduino este se divide en tres partes básicas: declaración de variables, setup() y loop().

- **Declaración de variables:** una variable puede delararse al inicio de un programa antes del void setup(), localmente dentro de funciones y a veces dentro de un bloque de declaración como por ejemplo un ciclo [7]. Si una variable es declarada al inicio de un programa, antes del setup(), la misma es considerada una variable global y puede utilizarse en cualquier función; por otra parte si es declarada dentro de una función o ciclo será considerada una variable local y sólo podrá utilizarse dentro de la función en que fue declarada. Los tipos de variables que existen en el lenguaje de programación son:

• Int: los enteros se usan para guardar datos numéricos sin puntos decimales. Almacenan un valor de 16 bits en el rango de 32767 a -32767, es decir que puede ser cualquier número dentro de ese rango.
• Unsigned int: posee las mismas funciones que "int" pero al ser una variable sin signo sus valores pueden variar entre 0 y 65535, al igual que "int" ocupa un espacio de 2 bytes.
• Long: el tipo "long" extiende el rango de valores enteros (sin decimales) a un valor de 32 bits, esto es un rango de 2147483647 a –2147483647, por lo que una variable long puede ser cualquier número entero dentro de este rango.
• Unsigned long: posee las mismas funciones que "long" con la diferencia de que no tiene signo negativo por lo que sus valores van desde 0 hasta 4294967295. Igualmente posee 32 bits.
• Byte: un byte ocupa 8 bits de la memoria de Arduino y puede representar cualquier número entero entre 0 y 255.
• Float: la variable float tiene mayor resolución que los enteros (int, long, byte) y se guarda como un valor de 32 bits en el rango desde 3.4028235E+38 a –3.4028235E+38. Esto quiere decir que es posible guardar un número decimal pero sólo en ese rango. Los números decimales (float) ocupan mucha memoria de Arduino. Usar decimales es mucho más lento que usar enteros, pues Arduino necesita más tiempo para realizar cálculos con éstos.
• Double: al igual que float almacena números decimales pero dispone de 64 bits de resolución.
• Char: estas variables se utilizan para guardar un único carácter y ocupan un byte.
• String: se utiliza para almacenar cadenas de caracteres.
• Boolean: este tipo de variables se utiliza para almacenar un valor lógico o booleano el cual puede ser "true" o "false".
• Arrays: si se desea guardar una colección de valores, entonces se debe utilizar una matriz. Todos los valores almacenados en una matriz se guardarán con un número índice, para acceder a cualquier valor se debe referenciar su número índice. Las matrices se declaran de igual modo que las variables (con el tipo, el nombre y los valores) pero a diferencia de una variable común, sus

valores son colocados dentro de llaves .

- **Void Setup:** al iniciar un programa en Arduino lo primero que se ejecuta es el void setup() y es la parte encargada de recoger la configuración. La función setup() se invoca una sola vez cuando el programa empieza. Se utiliza para inicializar los modos de trabajo de los pines, o el puerto serie. Debe ser incluido en un programa aunque no haya declaración que ejecutar. Así mismo se puede utilizar para establecer el estado inicial de las salidas de la placa [7]. La void setup() sólo se ejecuta una vez en el inicio y no se volverá a ejecutar hasta que Arduino se apague y vuelva a arrancar o se reinicie [8].
- **Void Loop:** esta función es el segundo paso escencial para un programa funcional [8], es la parte que contiene el programa que se ejecutará cíclicamente y se ejecuta después de la void setup(). La función loop() hace precisamente lo que sugiere su nombre, se ejecuta de forma cíclica, lo que posibilita que el programa esté respondiendo continuamente ante los eventos que se produzcan en la placa [7].

En cuanto a los comandos que se utilizan en la programación de Arduino se pueden mencionar los siguientes:

- Sintaxis básica:
  -Delimitadores: ; , {}
  -Comentarios: //, /* */
  -Cabeceras: #define, #include
  -Operadores aritméticos: $+$, $-$, $*$, $/$, %
  -Asignación: $=$
  -Operadores de comparación: $==$, $!=$, $<$, $>$, $<=$, $>=$
  -Operadores Booleanos: &&, ||, !
  -Operadores de acceso a punteros: *, &
  -Operadores de bits: &, |, , $<<$, $>>$
  -Operadores compuestos: $++$, $--$, $+=$, $-=$, $*=$, $/=$, &=, $|=$
- **Estructuras de control**
  -Condicionales: if, if...else, switch case

-Bucles: for, while, do... while

-Bifurcaciones y saltos: break, continue, return, goto

- **E/S digital**

  -Configurar pin digital: pinMode(pin, modo)

  -Escribir un valor en el pin digital: digitalWrite(pin, valor)

  -Leer un valor de entrada en un pin digital: digitalRead(pin)

- **E/S analógica**

  -Configurar el voltaje de referencia usado por la entrada analógica: analogReference(tipo)

  -Leer el valor del pin analógico: analogRead(pin)

  -Escribir valor en pin analógico (PWM): analogWrite(pin, valor)

- **Tiempo**

  -Muestra el tiempo en milisegundos desde que se inició el programa: millis()

  -Muestra el tiempo en microsegundos desde que se inicio el programa: micros()

  -Realizar un retraso en milisegundos: delay(ms)

  -Realizar un retraso en microsegundos: delayMicroseconds(microsegundos)

- **Comunicación por puerto serie**

  Las funciones de manejo del puerto serie deben ir precedidas de la palabra "Serial" aunque no necesitan ninguna declaración en la cabecera del programa. Por esto se consideran funciones base del lenguaje. Estas son las funciones para transmisión serial:

  -Configurar la tasa de bits en baudios: begin()

  -Desabilitar la comunicación serial: end()

  -Leer datos entrantes por el puerto serial: read()

  -Escribir un mensaje sin dejar un salto de línea: print()

  -Escribir un mensaje dejando un salto de línea: println()

  -Mostrar datos por el puerto serial en forma de bytes: write()

2.3. Protocolo SNMP

2.3.1. Definición

El protocolo SNMP es un protocolo de la capa de aplicación definido por Internet Architecture Board (IAB) en 1988 que facilita el intercambio de información de administración entre dispositivos de red. Es un protocolo abierto capaz de proveer una plataforma de administración común para varios dispositivos [9], utilizado para operaciones de monitoreo de redes; utiliza los puertos 161 y 162 de la capa transporte. SNMP está basado en el modelo de administrador/agente que consiste en un administrador, un agente, una base de datos de información de administración, los dispositivos administrados y el protocolo de red [9].

- **Administrador SNMP:** un administrador o sistema de administración es una entidad separada cuya responsabilidad es comunicarse con el agente SNMP implementado en los dispositivos de red. Son comúnmente referidos como NMS [10] y proporcionan el volumen de recursos de procesamiento y memoria requerido para la administración de la red. Las principales funciones del administrador son:
  - Consultar a los agentes.
  - Obtener respuestas de los agentes.
  - Fijar variables en los agentes.
  - Reconoce eventos asíncronos de los agentes.

- **Agente SNMP:** el agente es un programa que está incluido en el elemento de red. Un dispositivo debe habilitarse como agente y al hacerlo le permite recolectar información del dispositivo y le permite comunicarse con el administrador. Las funciones principales del administrador son:
  - Recolectar información de administración de su dispositivo.
  - Almacena y recupera información tal como se defina en el MIB.
  - Señala eventos al administrador.

• Actúa como proxy para algunos nodos que no se pueden administrar por SNMP.

- **Dispositivos administrados:** es un elemento de la red del que se requiere algún tipo de monitoreo y administración, por ejemplo un router, un switch o un servidor.

- **Base de datos de información de administración (MIB):** cada agente SNMP mantiene una base de datos de información que describe los parámetros del dispositivo administrado. El administrador SNMP usa esta base de datos para pedir información específica sobre el dispositivo y así mismo traduce la información como la necesita el sistema de administración. Esta base de datos compartida entre administrador y agente es lo que se conoce como (Management Information Base). El MIB está organizado en una estructura de árbol con variables individuales, como algún estado o una descripción, siento estas representadas como las hojas en las ramas. Para distinguir de manera única cada una de estas variables se hace uso de un identificador de objeto (OID) que no es más que una etiqueta numérica larga, de esta manera se identifica cada variable en el MIB y en los mensajes SNMP [9]. Un OID está conformado por una serie de números enteros separados entre sí por puntos, donde cada número representa una rama del árbol MIB. Por ejemplo la OID "sysDescr", utilizada para acceder a una breve descripción de un host, se identifica con el número .1.3.6.1.2.1.1.1 en la MIB, y ambas formas son equivalentes. En la figura 2.20 se puede observar una representación de este concepto [11].

2.3.2. Modo de Operación

Todos los dispositivos compatibles con SNMP contienen un MIB que consiste en información de atributos válidos de un dispositivo. Algunos atributos en la MIB son fijos, mientras que otros son valores dinámicos calculados por el NMS. La aplicación de gestión SNMP junto con el PC en el que se ejecuta es el NMS. Esto proporciona la mayor parte de los recursos de procesamiento y memoria necesarios para la gestión de red. Por lo tanto, el NMS ejecuta aplicaciones que supervisan y

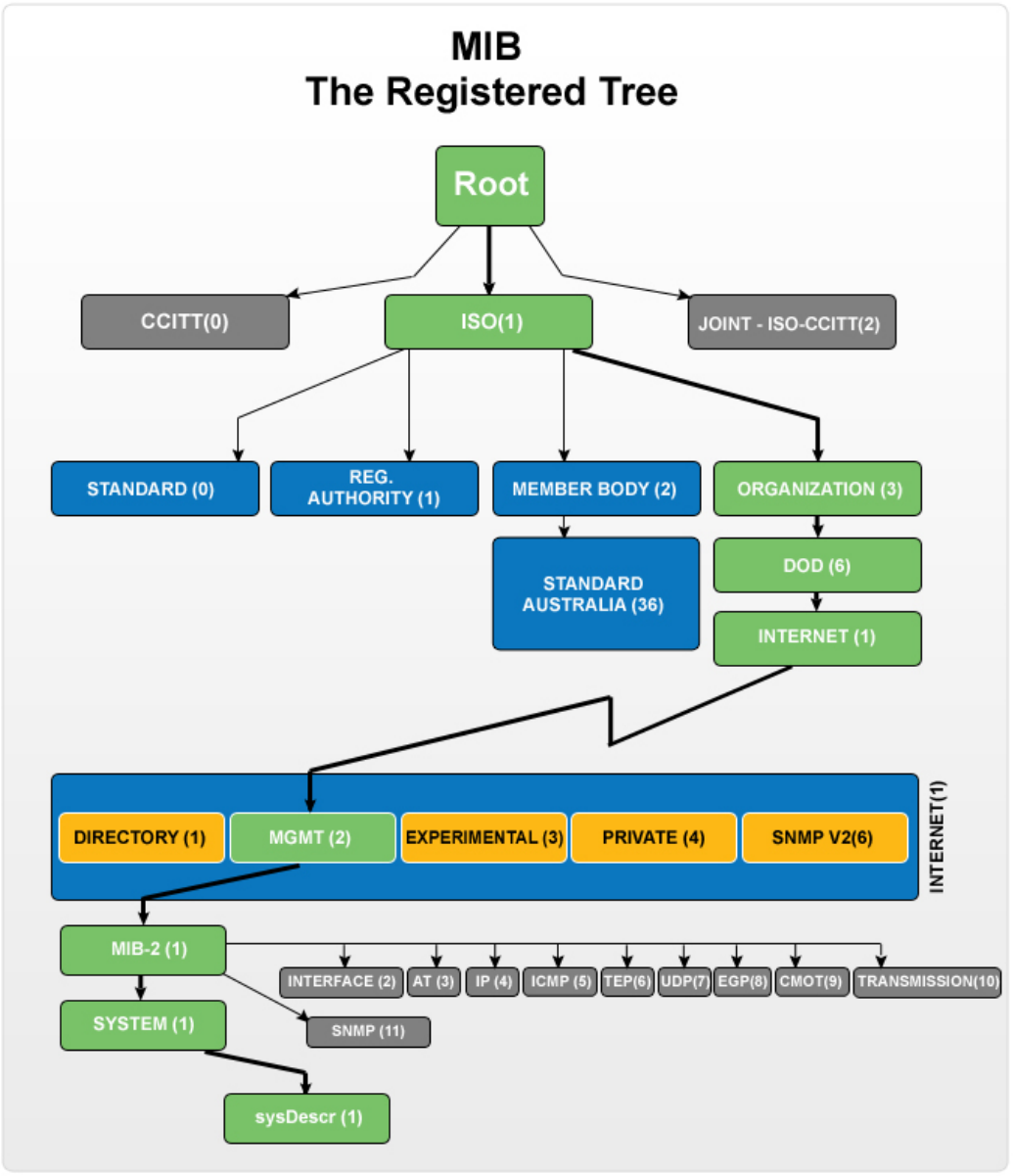


Figura 2.20: Diagrama de acceso al OID sysDescr en una árbol MIB

controlan los dispositivos administrados. El NMS utiliza comandos para leer datos que están almacenadosen la MIB del dispositivo. Los comandos "get" típicamente recuperan valores de datos, mientras que los comandos "set" típicamente inician alguna acción sobre el dispositivo [12]. Existe un formato PDU estándar para cada una de las siguientes operaciones SNMP [10]:

- Get.
- Get-next.
- Get-bulk (SNMPv2 y SNMPv3).
- Set.
- Get-response.
- Trap.
- Notification (SNMPv2 y SNMPv3).
- Inform (SNMPv2 y SNMPv3).
- Report (SNMPv2 y SNMPv3).

Capítulo III

Procedimiento Metodológico

3.1. Etapas de la Investigación

Para la realización de este proyecto se ha optado por separar las labores que involucra en etapas para mantener un orden y una facilidad en el desarrollo del mismo. Estas etapas son las siguientes:

3.1.1. Fase 1: Evaluación

- **Actividad A: Evaluar las instalaciones del nodo de la facultad de ingeniería de la Universidad de Carabobo para determinar los elementos disponibles** Al realizarse una visita al nodo se pudo comprobar que el prototipo que se había diseñado anteriormente [1] estaba en desuso y que presentaba elementos que podían reutilizarse en este nuevo prototipo, comprobando la factibilidad de realizar el prototipo de una manera rentable. Igualmente se verificó que las condiciones del nodo son aptas para la implementación del prototipo. Entre los elementos disponibles del prototipo anterior se planteó reutilizar un sensor de humo y un switch de contacto magnético, así como aprovechar los sensores de presión que trae incorporado el aire acondicionado que funciona en el nodo. Aparte, REDIUC colocó a disposición de los tesistas un detector de calor para ser incorporado al prototipo final.

3.1.2. Fase 2: Selección y Adquisición

- **Actividad B: Selección y adquisición de los sensores necesarios para la medición**

  En este punto ya se han definido las variables que han de ser sensadas y se comienza un proceso de documentación de los sensores disponibles y los sensores aún no adquiridos que existen en el mercado para su selección final, basados en nuestra capacidad de adquirilos, sus precios, su facilidad de uso/-programación y su capacidad de funcionar bajo las condiciones a las que van a ser sometidos. Durante el desarrollo de la Actividad A se pudo verificar que existen elementos rescatables del prototipo previo [1] y que pueden ser reutilizados en este nuevo prototipo, permitiendo una implementación más económica. Entre los sensores que se pudieron rescatar y adquirir para reutilizarse se encuentran:

  -**Detector iónico de humo SOVICA 1800-S:** diseñado para localizar las manifestaciones de humo cuando se produce incendio y enviar una señal de alerta a una central de incendio. Alimentación entre 10 y 32 voltios DC por medio de dos de sus cinco cables. Posee una sensibilidad de 1.5 % de humo.

  -**Detector térmico SOVICA 601-S:** diseñado para localizar cambios bruscos en la temperatura de un espacio determinado cuando se produce un incendio y se encarga de enviar una señal de alerta a una central de incendio. Alimentación entre 10 y 32 voltios DC por medio de dos de sus 5 cables. Posee un led rojo que se enciende al activarse y una sensibilidad de 57°C (137°F), funcionando en un área de cobertura de 37 m^2.

  -**Switch de contacto magnético:** utilizado en la puerta de acceso al nodo para notificar en caso de acceso no autorizado, funciona como un switch de dos piezas que abre/cierra según estas estén próximas entre sí. Estado normalmente abierto.

  -**Sensores de presión de aires acondicionados:** ya incorporados al aire acondicionado, son los presostatos de alta y baja que se accionan cuando la presión

en la etapa de succión y descarga del compresor no están en los niveles nominales de trabajo, en cuyo caso pueden llegar a dañarlo. Los presostatos utilizados en este proyecto trabajan en un rango de 25-80 psi el de baja y 250-350 psi el de alta.

Adicionalmente se planteó el monitoreo de la temperatura y humedad relativa de la habitación, por lo que buscó obtener nuevos sensores que permitieran la medicición de estas variables. Para lograr ambas mediciones se hizo obtención de una sensor **DHT11** de temperatura y humedad relativa a manera de simplificar el desarrollo del código y los costos. Entre las características de interés del sensor **DHT11** están:
-Salida digital y conexión de 3 pines, uno de alimentación, uno de tierra y uno de datos.
-Alimentación entre 3-5.5 voltios DC.
-Rango de medición entre 20-90 % de humedad relativa y 0-50°C de temperatura.
-Presición de ±5 % de humedad relativa y ±2°C de temperatura.
-Resolución de 1 % de humedad relativa y 1°C de temperatura.

- **Actividad C: Selección y adquisición de la tarjeta Arduino** Las variables y los sensores que van a usarse ya se han definido, por lo que hace falta la adquisición de una tarjeta Arduino que sea capaz de procesar la información proveniente de todos los sensores y cuyo coste no dificulte la implementación del prototipo o las réplicas que se generen basados en este. La tarjeta **Arduino UNO R3** es capaz de soportar la cantidad de sensores que se van a utilizar, es fácil de programar mediante su puerto USB incorporado y resulta más económica que otros modelos de tarjetas existentes. Por estos motivos y dado que se pudo acceder a esta tarjeta más fácilmente que a otros modelos se hizo la adquisición final una tarjeta **Arduino UNO R3**.

- **Actividad D: Selección y adquisición de la placa Ethernet del Arduino**
  Los datos que se vayan a adquirir con el Arduino deben transmitirse a través de la red para lograrse el monitoreo telemétrico; la mayoría de las placas Arduinos son incapaces de comunicarse directamente por un medio de red, por

lo que es necesario adquirir algún elemento compatible con el Arduino que permita adicionar esta funcionalidad al prototipo. Se decidió adquirir la placa **Arduino Ethernet Shield**, compatible con Arduino UNO y Arduino Mega mediante una librería que ya está integrada al software Arduino, por lo que es compatible con la placa Arduino ya adquirida. Algunas características importantes de esta placa Ethernet son: velocidad de conexión de 10/100Mb, voltaje de operación de 5 voltios (provenientes del Arduino) y puerto integrado para tarjeta micro SD que puede utilizarse para almacenar archivos al funcionar como servidor de red. Se muestra en la figura 3.1 una foto de una placa **Arduino Ethernet Shield**.

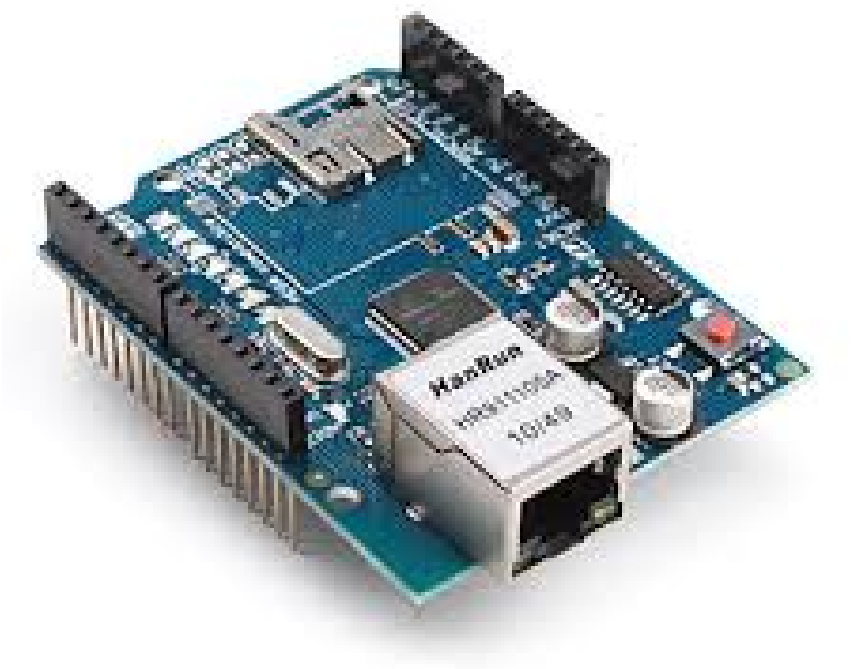


Figura 3.1: Placa Arduino Ethernet Shield

3.1.3. Fase 3: Diseño del Código de los Sensores

- **Actividad E: Diseñar un código Arduino individual para cada sensor**
  En esta etapa se realizaron los códigos de procesamiento de datos individuales para cada sensor de manera de poder comprobar su correcto funcionamiento con la tarjeta de manera de poder acoplarlos correctamente para realizar el código final en una etapa posterior. El software Arduino IDE es un software gratuito puesto a disposición de cualquiera en la página de Arduino, y es en este software donde se hizo el desarrollo de los códigos. En esta sección no se encuentra incorporada la comunicación SNMP, por lo que los códigos desarrollados permiten únicamente verificar el funcionamiento de los sensores mediante el Monitor Serial del software Arduino IDE. Se realizan entonces las conexiones apropiadas y el código para cada sensor:
  -Detector iónico de humo 1800-S: la empresa SOVICA, fabricante de este sensor, ofrece en su manual del usuario [13] un esquema de conexión típico del sensor a una central de incendios, tal como se muestra en la figura 3.2. El

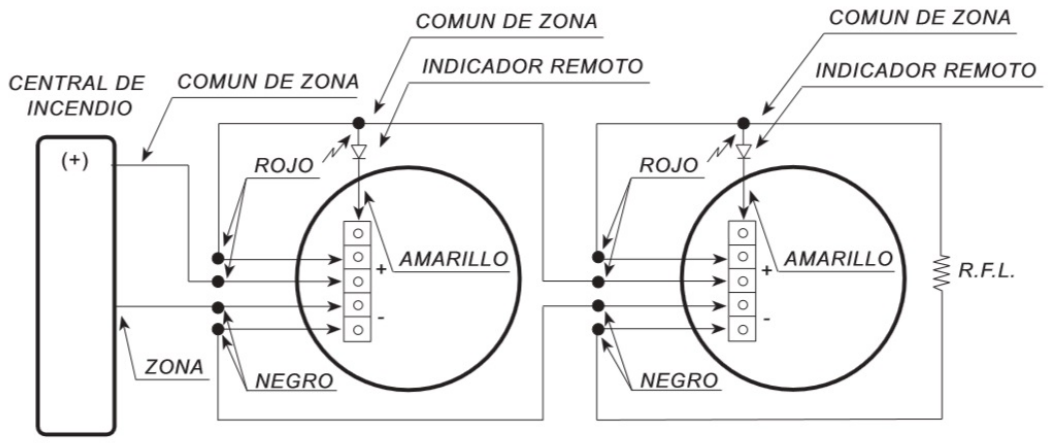


Figura 3.2: Esquema de conexión típico del sensor de humo 1800-S

sensor posee 5 cables de los cuales se disponen 2 pares para alimentar (dos terminales positivos y dos negativos) y el otro está disponible para obtener una señal que cambia al activarse una detección. El esquema muestra una conexión típica en la cual se conecta un dispositivo entre los terminales de alimentación y de detección. El sensor fue probado con una alimentación de

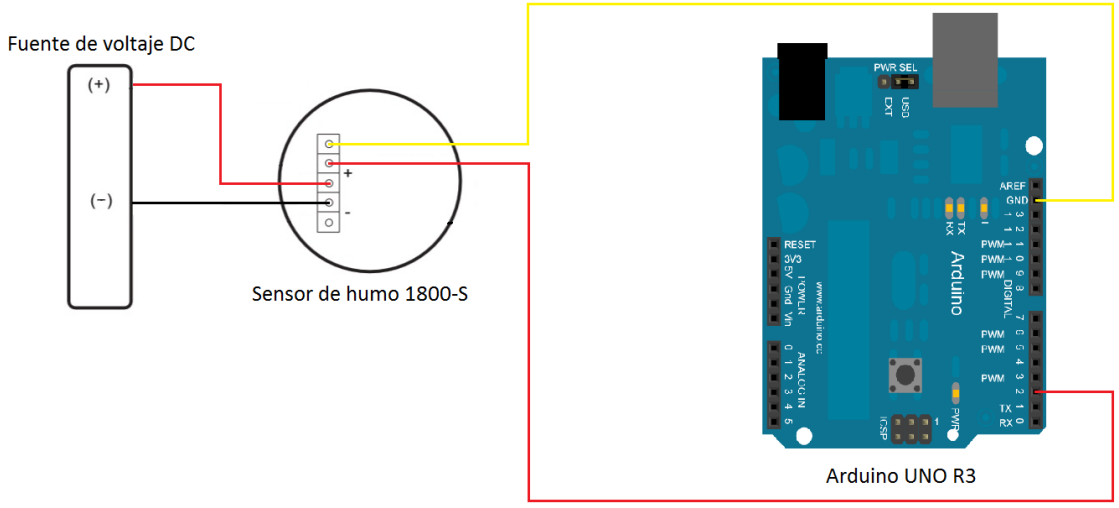


Figura 3.3: Esquema de conexión del Arduino y el detector de humo 1800-S

23.5 voltios y se midió el voltaje entre los terminales de alimentación y detección para un circuito abierto, resultando en un valor de 0 voltios al no haber detección de humo y de 4.8 voltios al haber detección de humo. El **Arduino UNO R3** está basado en el microcontrolador ATmega328, y según la hoja de especificaciones [14] el chip considera estos valores de voltaje en el rango de nivel lógico alto, por lo que se puede conectar el sensor a un puerto digital del Arduino como se muestra en la figura 3.3, por lo que se probó utilizando el siguiente código:

```
const int HumoPin = 2;   // La variable HumoPin indica el puerto digital
//donde se conecta el sensor de humo
int EstadoHumo = 1;  // La variable EstadoHumo almacena la lectura del puerto

void setup() {
        pinMode(HumoPin, INPUT);// Define el pin digital HumoPin como entrada
        Serial.begin(9600);   // Inicia la comunicacion serial con la
        //computadora a 9600 baudios
}

void loop() {
        EstadoHumo = digitalRead(HumoPin);   // Se chequea el nivel logico
```

```
        //del puerto donde esta conectado el sensor
        if (EstadoHumo == LOW) {
                Serial.println("Condicion normal");   // Si existe un cero
                //logico en el puerto se muestra en pantalla Condicion normal
        }
        else {
                Serial.println("PELIGRO!!!");   // Si existe un
                //uno logico en el puerto se muestra en pantalla PELIGRO!!!
        }
        delay(500);   // Cada medio segundo se va a comprobar el sensor
}
```

-Detector térmico 601-S: la empresa SOVICA, fabricante de este sensor, ofrece en su manual del usuario [15] un esquema de conexión típico del sensor a una central de incendios, que es idéntico al del sensor de humo que se muetra en la figura 3.2. El sensor posee 5 cables, de los cuales se tienen disponibles 2 pares para alimentar y el otro está disponible para obtener una señal que cambia al haber detección. Al igual que con el sensor iónico de humo este sensor fue probado con una alimentación de 23.5 voltios y fue conectado mediante los terminales de alimentación y detección a un par de pines del Arduino para comprobar que el nivel de voltaje que existe no excedieran los valores límites que soporta el microcontrolador ATmega328. En la figura 3.4 se muestra el esquema de conexión planteado para este sensor; tanto el esquema como el código planteado para este sensor son similares a los del sensor de humo. El código para probar este sensor es como sigue:

```
const int CalorPin = 3;   // La variable CalorPin indica el puerto digital
//donde se conecta el sensor de calor
int EstadoCalor = 1;// La variable EstadoCalor almacena la lectura del puerto

void setup() {
        pinMode(CalorPin, INPUT);   // Define el pin digital CalorPin como
        //entrada
        Serial.begin(9600);   // Inicia la comunicacion serial
        //con la computadora a 9600 baudios
}
```

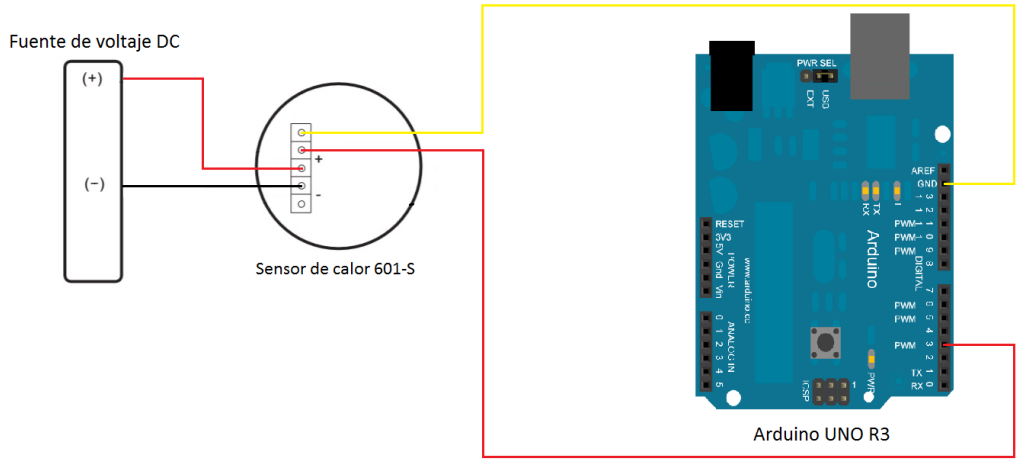


Figura 3.4: Esquema de conexión para el sensor de calor 601-S

```
void loop() {
        EstadoCalor = digitalRead(CalorPin);   // Se chequea el nivel logico
        //del puerto donde esta conectado el sensor
        if (EstadoHumo == LOW) {
                Serial.println("Condicion normal");   // Si existe un cero
                //logico en el puerto se muestra en pantalla Condicion normal
        }
        else {
                Serial.println("PELIGRO!!!");   // Si existe un uno logico
                //en el puerto se muestra en pantalla PELIGRO!!!
        }
        delay(500);   // Cada medio segundo se va a comprobar el sensor
}
```

-**Switch de contacto magnético:** este elemento es un reed switch de dos piezas que se cierra cuando ambas partes se encuentran próximas entre sí y se abre cuando estas se alejan. Para conectar este switch al Arduino se plantea el uso de una resistencia pull-up interna del Arduino, quedando el esquema de conexión como en la figura 3.5. Con el esquema planteado cuando el switch está cerrado el Arduino lee un 0 lógico y al estar abierto lee un 1 lógico. El código para probar el sensor es como sigue:

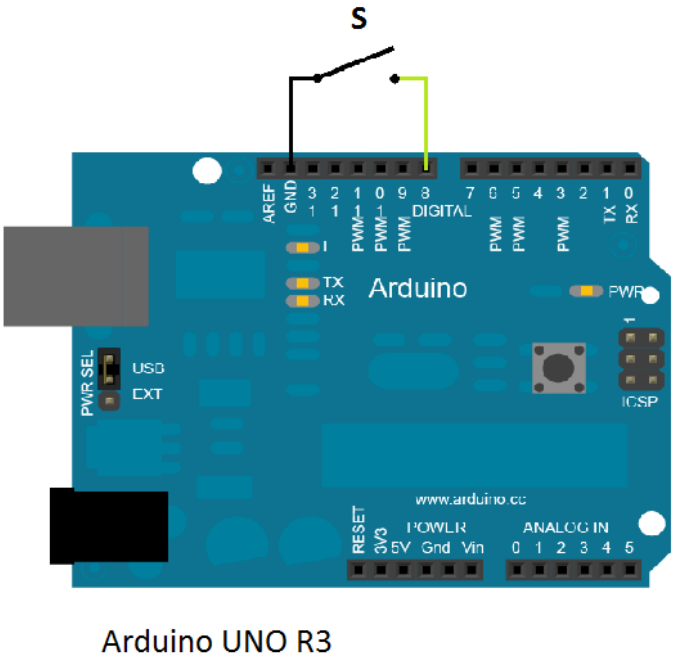


Figura 3.5: Esquema de conexión del sensor de contacto magnético al Arduino

```
const int PinMagnetico = 8;   // La variable PinMagnetico indica el puerto
//digital donde se conecta el sensor de humo
int EstadoMagnetico = 1;   // La variable EstadoMagnetico almacena la
//lectura del puerto
int UltimoEstadoMagnetico = digitalRead(PinMagnetico);   // La variable
//UltimoEstadoMagnetico almacena el ultimo estado leido

void setup() {
        pinMode(PinMagnetico, INPUT_PULLUP);   // Define el pin digital
        //PinMagnetico como entrada con resistencia pull-up interna
        Serial.begin(9600);   // Inicia la comunicacion serial con la
        //computadora a 9600 baudios
}

void loop() {
        EstadoMagnetico = digitalRead(PinMagnetico);   // Se chequea el nivel
```

```
        //logico del puerto donde esta conectado el sensor
        if (EstadoMagnetico != UltimoEstadoMagnetico) {
                if (EstadoMagnetico == LOW) {
                        Serial.println("Switch cerrado");   // Si se lee un
                        //cero logico en el puerto se muestra en pantalla
                        //Switch cerrado
                }
                else {
                        Serial.println("Switch abierto");   // Si se lee un
                        //uno logico en el puerto se muestra en pantalla
                        //Switch abierto
                }
                UltimoEstadoMagnetico = EstadoMagnetico;
        }
        delay(500);   // Cada medio segundo se va a comprobar el switch
}
```

-**Sensores de presión de aires acondicionados:** un aire acondicionado como el del nodo posee 2 presostatos que se accionan cuando la presión de entrada o de salida del compresor se encuentra fuera de los niveles normales de trabajo; estos presostatos son llamados de baja y de alta, y funcionan como switches que se abren y desenergizan el compresor cuando los niveles de presión no son adecuados. En la imagen 3.6 se puede observar un esquema simplificado de su conexión [16]; los presostatos conectan un voltaje AC de 24 voltios RMS al compresor. Cada presotato se sensa midiendo el voltaje en su par de termi-

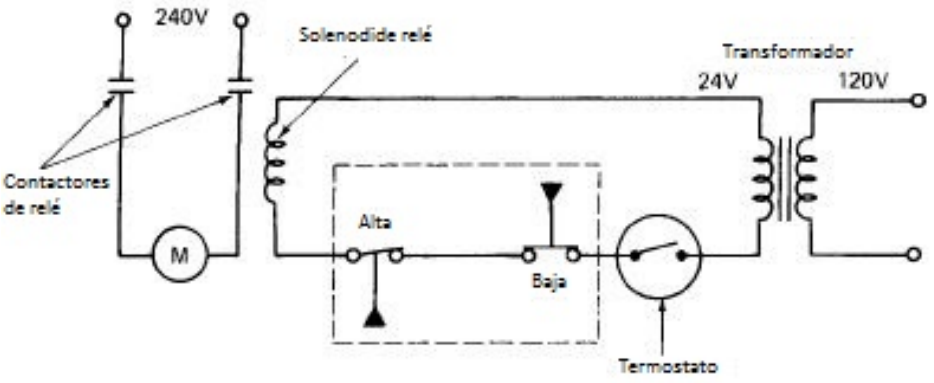


Figura 3.6: Esquema simplificado de conexión de los presostatos. Imagen traducida

nales, el cual resulta en 0 voltios al estar en condiciones correctas de trabajo y en 24 voltios AC al detectar una presión no adecuada; esta señal es la que ha de sensarse con el Arduino, la cual debe acondicionarse para trabajar en los niveles adecuados para que este pueda leerla. La señal ha de medirse como una señal digital, por lo que hay que realizar una conversión de la señal de 24 voltios AC a una señal DC de 5 voltios. Para lograrlo se debe colocar una etapa de rectificación y regulación para cada presostato, por lo que en la figura 3.7 se observa el circuito planteado para el acondicionamiento de la señal. A

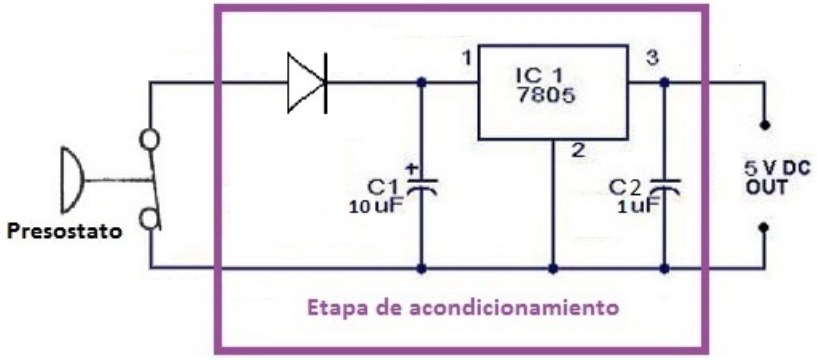


Figura 3.7: Circuito acondicionador de señal para los presostatos

la salida de la etapa acondicionadora se obtiene la señal digital lista para ser sensada por el Arduino, por lo que el esquema de conexión de los presostatos al Arduino quedaría como se muestra en la figura 3.8. El código desarrollado para comprobar el estado de los presostatos es como sigue:

```
const int PinPresionBaja = 6;   // La variable PinPresionBaja indica el puerto
//digital donde se conecta el presostato de baja
const int PinPresionAlta = 7;   // La variable PinPresionAlta indica el puerto
//digital donde se conecta el presostato de alta

void setup() {
        pinMode(PinPresionBaja, INPUT);   // Define el pin digital
        //PinPresionBaja como entrada
        pinMode(PinPresionAlta, INPUT);   // Define el pin digital
        //PinPresionAlta como entrada
```

```
        Serial.begin(9600);   // Inicia la comunicacion serial con la
        //computadora a 9600 baudios
}

void loop() {
        if (digitalRead(PinPresionBaja) == HIGH) {
                Serial.println("Falla en la presion de baja");   // Si se lee
                //un uno logico en el puerto se muestra en pantalla que hay
                //falla de presion baja
        }
        if (digitalRead(PinPresionAlta) == HIGH) {
                Serial.println("Falla en la presion de alta");   // Si se lee
                //un uno logico en el puerto se muestra en pantalla que hay
                //falla de presion alta
        }
        delay(500);   // Cada medio segundo se comprueban los presostatos
}
```

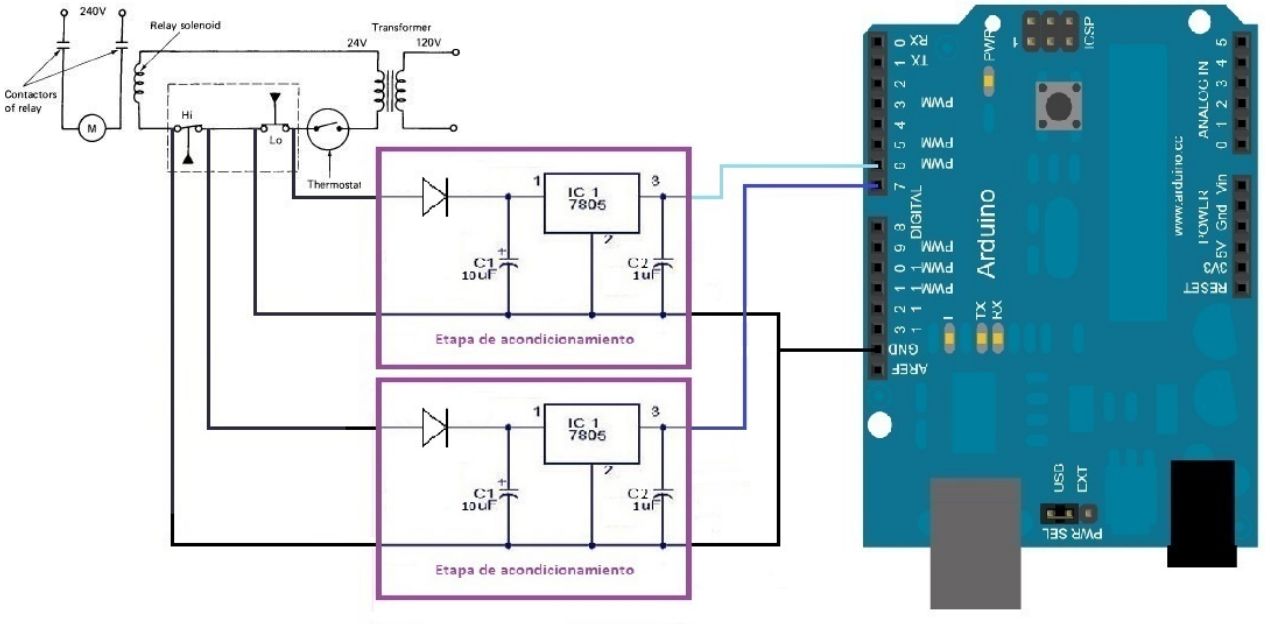


Figura 3.8: Esquema de conexión de los presostatos al Arduino

-**Sensor de temperatura y humedad DHT11:** este sensor consta de 3 pines (aunque algunos modelos constan de 4) y en su manual de usuario [17] presentan un esquema de conexión típico, que se muestra en la figura 3.9. Para hacer la prueba del sensor se hace una conexión entre el sensor y el Arduino como se muetra en la figura 3.10 y se carga el Arduino con el siguiente código:

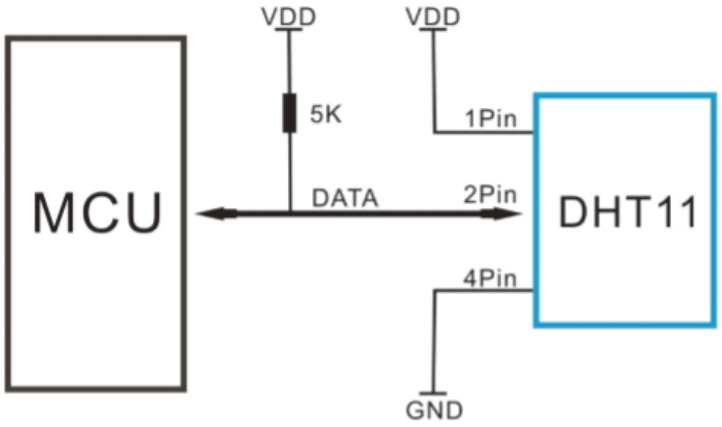


Figura 3.9: Esquema de conexión típico del sensor DHT11

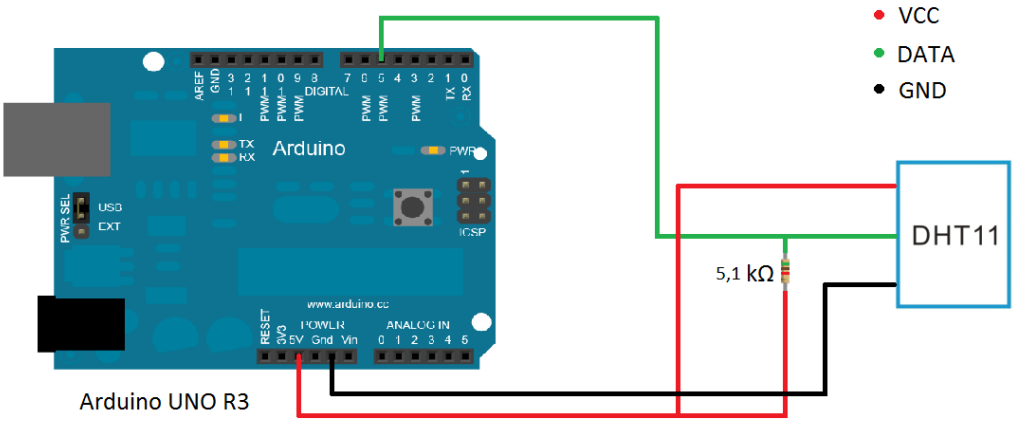


Figura 3.10: Esquema de conexión del sensor DHT11 al Arduino

```
#include "DHT.h"   // Se incluye la libreria de los sensores DHT

DHT dht(5,DHT11);   // Se define que por el puerto digital 5 se conectara un
```

```
//sensor DHT11

void setup() {
        Serial.begin(9600);   // Se inicia la comunicacion serial con la
        //computadora a 9600 baudios
        dht.begin();   // Inicia el sensor
        delay(1000); // Se espera 1 segundo antes de empezar las lecturas
}

void loop() {
        int h = dht.readHumidity();   // Se lee la humedad
        int t = dht.readTemperature();   // Se lee la temperatura
        Serial.print("Humedad: ");
        Serial.print(h);   // Se muestra en pantalla el valor de la humedad
        Serial.println("%");
        Serial.print("Temperatura: ");
        Serial.print(t); // Se muestra en pantalla el valor de la temperatura
        Serial.println(" grados");
        Serial.println();
        delay(500);   // Se hacen lecturas cada medio segundo
}
```

NOTA: la librería de los sensores DHT no viene incluída en las librerías por defecto del compilador Arduino, debe descargarse de internet y agregarse a las bilbiotecas del compilador.

- **Actividad F: Realizar pruebas individuales con los sensores y Arduino**
  Esta actividad pone a prueba el esquema de conexión y el código de cada sensor, planteados en la actividad F; esto se realiza con la finalidad de verificar el correcto funcionamiento de los códigos y elementos involucrados, para su posterior acoplamiento en el prototipo final. Esta verificación se hace mediante el Monitor Serial del software Arduino IDE, aunque puede realizarse con cualquier herramienta que reciba la información del puerto serial donde se conecte el prototipo, como por ejemplo HyperTerminal. Esta actividad se realiza con cada uno de los sensores y se comentan los resultados que se observan para cada uno:
  -**Detector iónico de humo 1800-S:** el código para este sensor funciona leyendo

el estado del detector de humo cada medio segundo y mostrando en pantalla un mensaje de ”Condicion normal´´ o ”PELIGRO!!´´ según el estado del detector. Al cargar el Arduino con el código correspondiente a este sensor se puede observar de resultado una imagen como la 3.11.

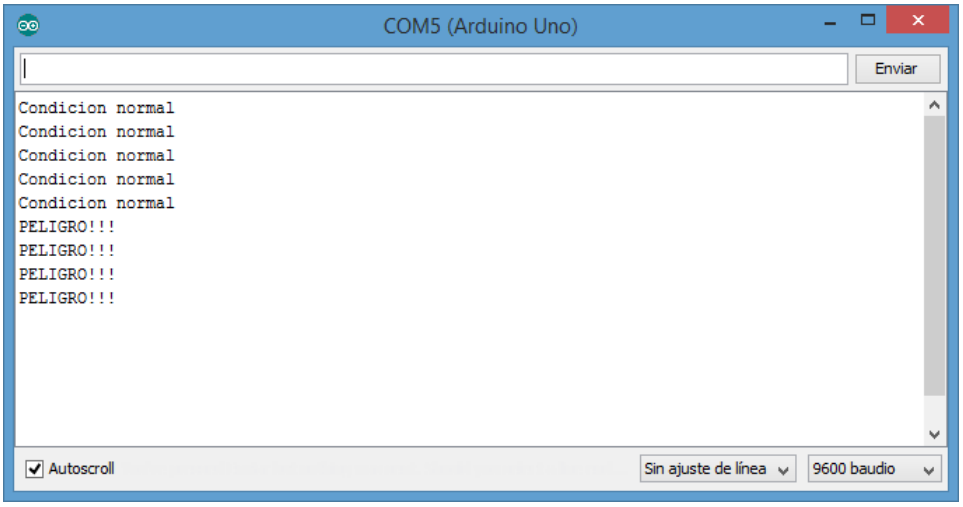


Figura 3.11: Lecturas seriales del computador al probar el Arduino con el detector iónico de humo

-**Detector térmico 601-S:** este código funciona de manera idéntica al código del detector de humo, se comprueba el estado del sensor térmico cada medio segundo y se imprime en pantalla un mensaje de ”Condicion normal´´ o ”PELIGRO!!´´ según su estado, tal como se ve en la imagen 3.12.

-**Switch de contacto magnético:** el código para el switch magnético funciona comprobando cada medio segundo el cambio de estado del switch e imprimiendo en pantalla un único mensaje de ”Switch abierto´´ o ”Switch cerrado´´ según el estado al que cambió. Una imagen como la 3.13 debería observarse al poner en funcionamiento el Arduino con el código de este sensor.

-**Sensores de presión de aires acondicionados:** el código de los presostatos funciona comprobando el estado de cada uno e imprimiendo en pantalla cada medio segundo el mensaje ”Falla en la presion de baja´´ o ”Falla en la presion de alta´´ según el presostato que detecta presión distinta a la nominal. La imagen 3.14 muestra el resultado de poner este código a prueba.

-**Sensor de temperatura y humedad DHT11:** para el sensor DHT11 el código

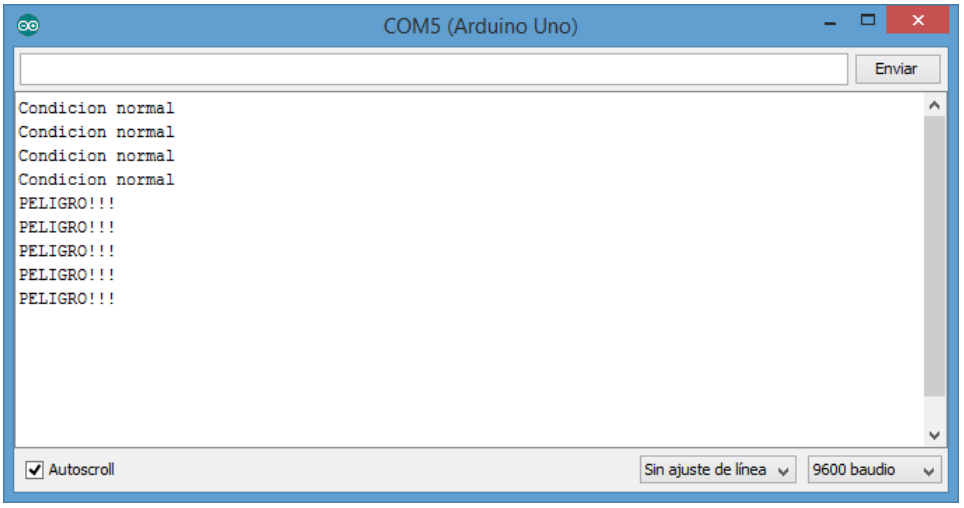


Figura 3.12: Lecturas seriales del computador al probar el Arduino con el detector térmico

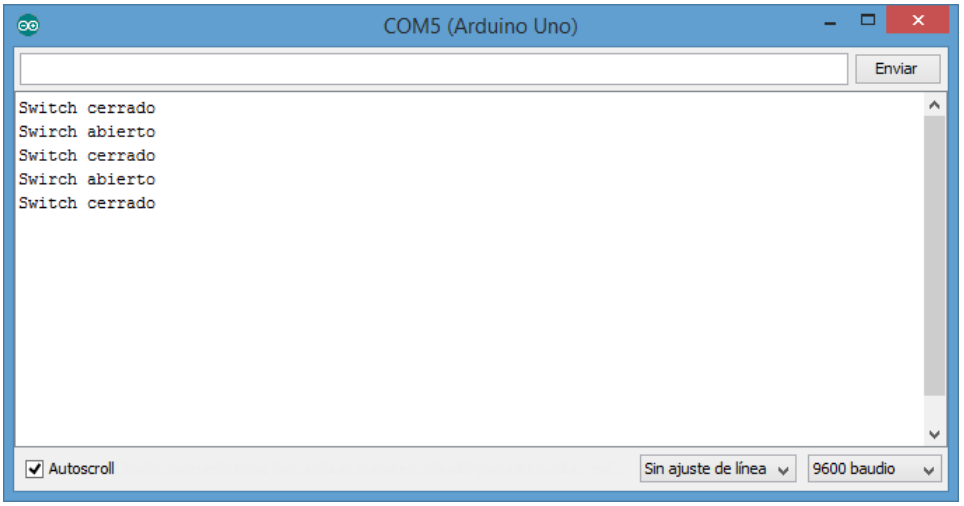


Figura 3.13: Lecturas seriales del computador al probar el Arduino con el switch de contacto magnético

funciona accediendo a las lecturas de temperatura y humedad del sensor e imprimiéndolas en pantalla de manera cíclica cada medio segundo. El resultado de utilizar este código se puede observar en la imagen 3.15.

- **Actividad G: Diseñar un código global en Arduino para todos los sensores** En esta actividad se procede a integrar todos los sensores al Arduino para poner a prueba el funcionamiento conjunto de ellos en un esquema de conexión como el que se muestra en la imagen 3.16. El código que se desarrolla

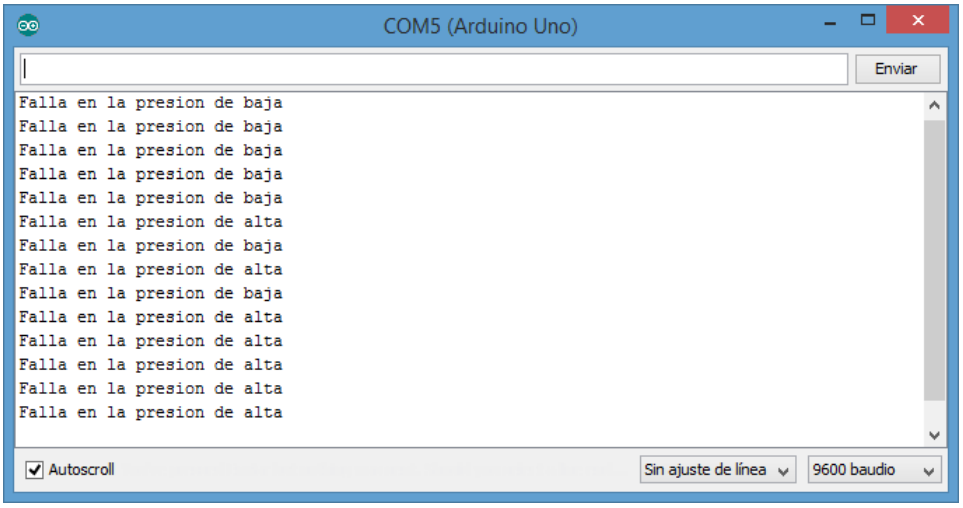


Figura 3.14: Lecturas seriales del computador al probar el Arduino con los presostatos

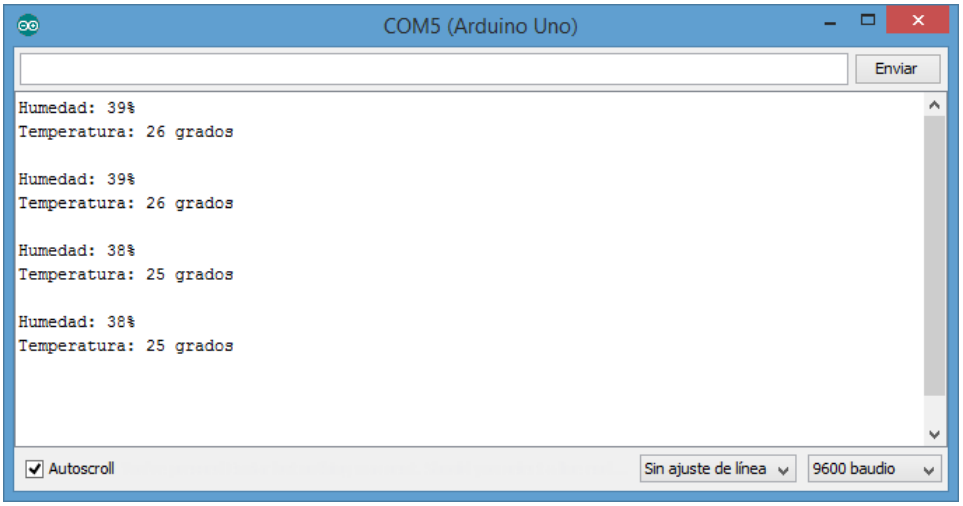


Figura 3.15: Lecturas seriales del computador al probar el Arduino con el sensor DHT11

no posee comunicación SNMP, por lo que sólo sirve para verificar el funcionamiento de los sensores y la correcta captura de las mediciones para que en una etapa posterior se pueda enviar esta información mediante el protocolo definido. Este código resulta en una combinación de los códigos individuales de cada sensor desarrolado en la actividad previa, por lo que permite verificar mediante comunicación serial el funcionamiento conjunto de los sensores y resulta como se muestra a continuación:

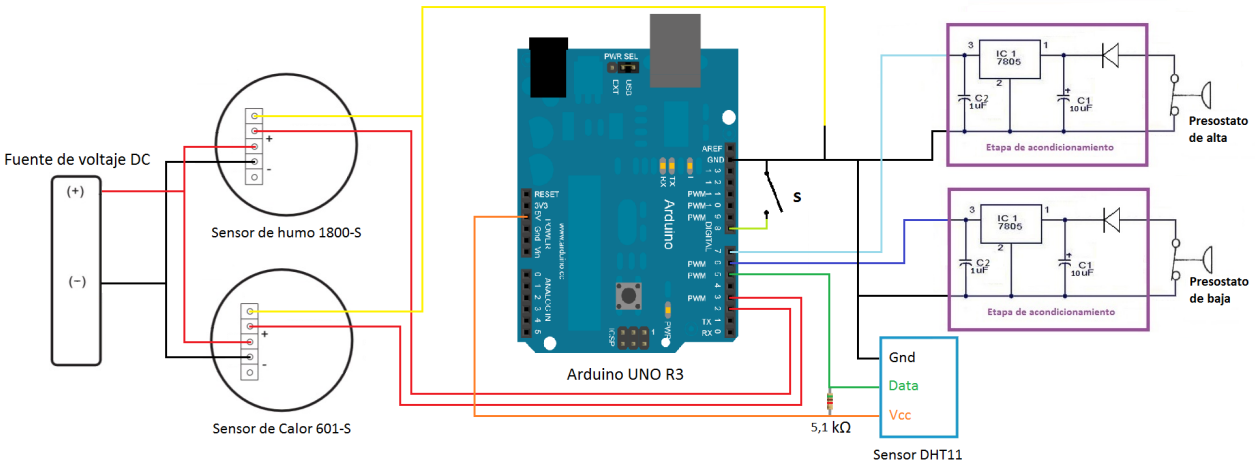


Figura 3.16: Esquema de conexión global de los sensores

```
#include "DHT.h"   // Se incluye la libreria de los sensores DHT

const int HumoPin = 2;   // La variable HumoPin indica el puerto digital
//donde se conecta el sensor de humo
int EstadoHumo = 1;  // La variable EstadoHumo almacena la lectura del puerto
const int CalorPin = 3;   // Pin analogico donde se lee el estado del sensor
//de calor
int EstadoCalor = 1; //La variable EstadoCalor almacena la lectura del puerto
DHT dht(5,DHT11);   // Se define que por el puerto digital 5 se conectara un
//sensor DHT11
const int PinPresionBaja = 6;  // La variable PinPresionBaja indica el puerto
//digital donde se conecta el presostato de baja
int EstadoPresionBaja = 1;     // La variable EstadoPresionBaja almacena la
//lectura del puerto
const int PinPresionAlta = 7;  // La variable PinPresionAlta indica el puerto
//digital donde se conecta el presostato de alta
int EstadoPresionAlta = 1;     // La variable EstadoPresionAlta almacena la
//lectura del puerto
const int PinMagnetico = 8;   // La variable PinMagnetico indica el puerto
//digital donde se conecta el switch
int EstadoMagnetico = 1;   // La variable EstadoMagnetico almacena la lectura
//del puerto
```

```
int UltimoEstadoMagnetico = digitalRead(PinMagnetico);   // La variable
//UltimoEstadoMagnetico almacena el ultimo estado leido

void setup() {
        pinMode(HumoPin, INPUT); //Define el pin digital HumoPin como entrada
        pinMode(CalorPin, INPUT); // Define el pin digital CalorPin como
        //entrada
        pinMode(PinPresionBaja, INPUT);   // Define el pin digital
        //PinPresionBaja como entrada
        pinMode(PinPresionAlta, INPUT);   // Define el pin digital
        //PinPresionAlta como entrada
        pinMode(PinMagnetico, INPUT_PULLUP);   // Define el pin digital
        //PinMagnetico como entrada con resistencia pull-up interna
        Serial.begin(9600);   //Se comienza la comunicacion serial con la
        //computadora a 9600 baudios
        dht.begin();   // Inicia el sensor de humedad
        delay(1000);   // Se espera 1 segundo antes comenzar las lecturas
}

void loop() {
        int h = dht.readHumidity();   // Se lee la humedad
        int t = dht.readTemperature();   // Se lee la temperatura
        EstadoHumo = digitalRead(HumoPin);   // Se chequea el nivel logico
        //del puerto donde esta conectado el sensor de humo
        EstadoCalor = digitalRead(CalorPin);   // Se chequea el nivel logico
        //del puerto donde esta conectado el sensor de calor
        EstadoPresionBaja = digitalRead(PinPresionBaja);     // Se chequea el
        //nivel logico del puerto donde esta conectado el presostato de baja
        EstadoPresionAlta = digitalRead(PinPresionAlta);     // Se chequea el
        //nivel logico del puerto donde esta conectado el presostato de alta
        if (EstadoHumo == HIGH || EstadoCalor == HIGH) {
                Serial.println("PELIGRO DE INCENDIO!!!");   // Si el detector
                //de humo o el de calor detecta algo se muestra en pantalla
                //PELIGRO!!!
        }
        Serial.print("Humedad: ");
        Serial.print(h);   // Se muestra en pantalla el valor de la humedad
        Serial.println("%");
        Serial.print("Temperatura: ");
        Serial.print(t); // Se muestra en pantalla el valor de la temperatura
```

```
        Serial.println(" grados");
        if (EstadoPresionBaja == HIGH) {
                Serial.println("Falla en la presion de baja");   // Si se lee
                //un uno logico en el puerto se muestra en pantalla que hay
                //falla de presion baja
        }
        if (EstadoPresionAlta == HIGH) {
                Serial.println("Falla en la presion de alta");   // Si se lee
                //un uno logico en el puerto se muestra en pantalla que hay
                //falla de presion alta
        }
        EstadoMagnetico = digitalRead(PinMagnetico);   // Se chequea el nivel
        //logico del puerto donde esta conectado el sensor
        if (EstadoMagnetico != UltimoEstadoMagnetico) {
                if (EstadoMagnetico == LOW) {
                        Serial.println("Switch cerrado");   // Si se lee un
                        //cero logico en el puerto se muestra en pantalla
                        //Switch cerrado
                }
                else {
                        Serial.println("Swirch abierto");   // Si se lee un
                        //uno logico en el puerto se muestra en pantalla
                        //Switch abierto
                }
                UltimoEstadoMagnetico = EstadoMagnetico;
        }
        Serial.println();
        delay(500);   // Se hacen lecturas cada medio segundo
}
```

- **Actividad H: Realizar pruebas con los sensores y el Arduino en un ambiente controlado y comprobar funcionalidad**

  Luego de desarrollar un esquema de conexión y un código global para todos los sensores es necesario comprobar que estos funcionan correctamente con el Arduino. Como se estableció en la actividad previa el código desarrollado para esta actividad no incluye comunicación SNMP y sirve para verificar únicamente el funcionamiento conjunto de los sensores mediante comunicación serial. El código funciona de manera similar a como funcionan los códigos

anteriores, cada medio segundo se verifica el estado de los sensores, si los detectores de humo o calor detectan incendio se envía el mensaje ”PELIGRO DE INCENDIO!!!´´, luego se envían las lecturas de humedad o temperatura, un mensaje ”Falla en la presion de baja´´ o ”Falla en la presion de alta´´ según si se detecta una presión diferente a la nominal en el presostato de baja o alta, y finalmente se envía un mensaje de ”Switch cerrado´´ o ”Switch abierto´´ de haberse cerrado o abierto el switch magnético. El Arduino al ejecutar el programa muestra un resultado como el de la figura 3.17 al utilizarse el Arduino IDE.

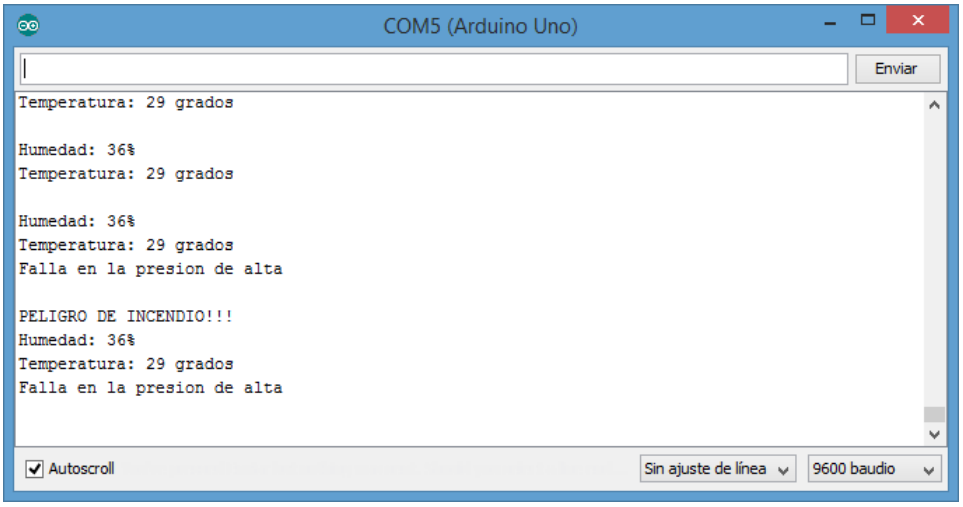


Figura 3.17: Lecturas seriales del computador al probar el Arduino con todos los sensores simultáneamente

3.1.4. Fase 4: Diseño del Código de Comunicación SNMP

- **Actividad I: Diseñar un código basado en Arduino para la placa Ethernet adquirida que permita entregar las mediciones a un servidor SNMP**
  En este punto las variables a monitorear son capturadas correctamente por el Arduino, por lo que se desarrolla un código que permita entregar estas mediciones cómodamente al administrador mediante protocolo SNMP. El protocolo SNMP versión 1 posee cinco mensajes: *get*, *set*, *walk*, *response* y *trap*; de estos, sólo 2 de ellos son generados por el agente SNMP que en nuestro caso vendría a ser el prototipo de monitoreo. Un mensaje response es generado

por el agente SNMP en respuesta a un mensaje *get*, *set* o *get-next*, mientras que un mensaje *trap* lo genera el agente al ocurrir un evento que dispare la trampa. Dado que muchas de las variables que se monitorean pueden obtener sólo uno de dos posibles estados (detección o no de humo, puerta abierta/cerrada, presostato abierto/cerrado) se decidió que el prototipo debía enviar información de estos eventos mediante mensajes tipo *trap* cada vez que ocurría uno de los eventos de interés, a la vez que pueden ser consultados sus estados con solicitudes *get*; de manera similar variables que pueden tomar cualquier valor en un rango amplio y que van a ser graficadas (como lo son la temperatura y la humedad) responderán a mensajes tipo *get* también.

Para lograr tal objetivo se hizo uso de una librería llamada Agentuino, capaz de otorgarle al Arduino la capacidad de responder a mensajes tipo *get*, *set* y *get-next*, generando un mensaje tipo response según el caso, y ya que la librería no soporta la generación de mensajes tipo *trap*, fue modificada y se le agregaron las funciones necesarias para enviar las trampas que definen cada evento.

La librería Agentuino consta de 2 tipos de archivos, un archivo de extensión .h y un archivo de extensión .cpp, y es en el archivo.cpp donde se crearon las funciones que respondieran con un mensaje tipo trampa y las variables asociadas a estas, mientras que en el archivo .h se hicieron unas modificaciones para agregar la funciones definidas en el archivo.cpp. Las funciones creadas en el archivo .cpp corresponden a una función para cada trampa que fuese a generarse, construida mayormente byte por byte según el formato de una trampa SNMP [18], y dejando algunos campos variables para poder modificarse. A continuación se puede observar una de estas funciones, llamada TrampaIncendio y anexada a la librería Agentuino, la cual transmite el mensaje "PELIGRO DE DETECCIÓN DE HUMO" al ser invocada:

```
void AgentuinoClass::TrampaHumo(byte RemIP[4], byte TrapIP[4],
uint32_t Tiempo) {
```

```
char Mensaje[] = "PELIGRO DE DETECCION DE HUMO";     // Mensaje a
//enviarse en la trampa
byte TipoyLongitud[2]={48, 63+strlen(Mensaje)+strlen(_trapCommName)};
byte Version[3] = {2, 1, 0};   // Defined version 1
byte TamanhoComunidad[2] = {4, strlen(_trapCommName)};
byte TipoSNMP[4] = {164, 130, 0, 54+strlen(Mensaje)};
byte OID[11] = {6, 9, 43, 6, 1, 4, 1, 225, 125, 131, 118}
byte IPdefinida[2] = {64, 4};
byte TipoTrampa[3] = {2, 1, 6};
byte SpecificTrapNumber[3] = {2, 1, 1};
byte TipoTiempo[2] = {67, 4};
byte VarBind[4] = {48, 130, 0, 21+strlen(Mensaje)};
byte VarBind1[4] = {48, 130, 0 , 17+strlen(Mensaje)};
byte OID1[15] = {6,12,43,6,1,4,1,225,125,131,118,1,1,3,0};
byte Value1[2] = {4, strlen(Mensaje)};

int i=0,k=1,temp;
byte suma = 0;
uint32_t quotient;
quotient = Tiempo;
byte hexadecimalNumber[4] ={0, 0, 0, 0};
while (quotient!=0)
{
        temp = quotient % 16;
        if (k == 1)
        {
                suma = temp;
                k = 2;
        }
        else
        {
                suma = suma + temp*16;
                hexadecimalNumber[3-i] = suma;
                i = i + 1;
                k = 1;
        }
        quotient = quotient / 16;
}
if (k == 2)
{
```

```
                hexadecimalNumber[3-i] = suma;
        }

        Udp.beginPacket(RemIP, SNMP_DEFAULT_TRAPPORT);
        Udp.write(TipoyLongitud,2);
        Udp.write(Version,3);
        Udp.write(TamanhoComunidad,2);
        Udp.write(_trapCommName,strlen(_trapCommName));
        Udp.write(TipoSNMP,4);
        Udp.write(OID,11);
        Udp.write(IPdefinida,2);
        Udp.write(TrapIP, 4);
        Udp.write(TipoTrampa,3);
        Udp.write(SpecificTrapNumber,3);
        Udp.write(TipoTiempo,2);
        Udp.write(hexadecimalNumber,4);
        Udp.write(VarBind,4);
        Udp.write(VarBind1,4);
        Udp.write(OID1,15);
        Udp.write(Value1,2);
        Udp.write(Mensaje,strlen(Mensaje));
        Udp.endPacket();
}
```

La función previa transmite una trampa SNMP al haber detección de humo y de manera similar se creó una función para cada evento generador de trampa, cada una con su propio mensaje, y cada una requiriendo que se le especifique la dirección IP de destino de la trampa, la IP que contendrá la trampa como de IP de origen y el tiempo que lleva el arduino encendido; estas variables se definen en el código principal del prototipo. Se crearon seis funciones de trampa para el prototipo y son las siguientes:

- Agentuino.TrampaHumo: trampa para informar que hubo detección de humo con el mensaje "PELIGRO DE DETECCIÓN DE HUMO".
- Agentuino.TrampaCalor: trampa para informar que hubo detección de calor con el mensaje "PELIGRO DE DETECCIÓN DE CALOR".

- Agentuino.TrampaPuertaAbierta: trampa para informar que la puerta del nodo ha sido abierta con el mensaje "Puerta abierta".
- Agentuino.TrampaPuertaCerrada: trampa para informar que la puerta del nodo ha sido cerrada con el mensaje "Puerta cerrada".
- Agentuino.TrampaPresionBaja: trampa para informar que hay una falla en la presión baja con el mensaje "Falla en la presión baja".
- Agentuino.TrampaPresionAlta: trampa para informar que hay una falla en la presión alta con el mensaje "Falla en la presión alta".

Con las funciones de trampa agregadas a la librería se pudo generar un código que permitiese actuar al prototipo como un agente SNMP con versión 1 del protocolo. Este código combina el código previo de los sensores y una modificación del código ejemplo de Agentuino, convirtiendo al prototipo en un sistema de monitoreo remoto que funciona con el protocolo SNMP. El código utilizado para habilitar comunicación SNMP es muy extenso, por lo que no fue incorporado en este trabajo, sin embargo a continuación se muestra la función loop, la cual escucha solicitudes SNMP y monitorea los eventos de disparo de trampas:

```
void loop() {
        EstadoHumo = digitalRead(HumoPin);     // Se chequea el nivel logico
        //del puerto donde esta conectado el sensor de humo
        EstadoCalor = digitalRead(CalorPin);     // Se chequea el nivel
        //logico del puerto donde esta conectado el sensor de calor
        EstadoPresionBaja = digitalRead(PinPresionBaja);     // Se chequea el
        //nivel logico del puerto donde esta conectado el presostato de baja
        EstadoPresionAlta = digitalRead(PinPresionAlta);     // Se chequea el
        //nivel logico del puerto donde esta conectado el presostato de alta
        EstadoMagnetico = digitalRead(PinMagnetico);     // Se chequea el
        //nivel logico del puerto donde esta conectado el switch magnetico
        if (EstadoHumo == HIGH) {
                if ( millis() - tiempoHumo >= 5000 ) {     // Cada 5
                //segundos se envia la trampa de humo
                        Agentuino.TrampaHumo(RemoteIP, ip, locUpTime);
                        tiempoHumo = millis();
```

```
        }
    } else {
        tiempoHumo = 0;
        delay(50);
        locUpTime = locUpTime + 5;
    }
    if (EstadoCalor == HIGH) {
        if ( millis() - tiempoCalor >= 5000 ) {     // Cada 5
        //segundos se envia la trampa de calor
            Agentuino.TrampaCalor(RemoteIP, ip, locUpTime);
            tiempoCalor = millis();
        }
    } else {
        tiempoCalor = 0;
        delay(50);
        locUpTime = locUpTime + 5;
    }
    if (EstadoPresionBaja == HIGH) {
        if ( millis() - tiempoPresionBaja >= 5000 ) {     // Cada 5
        //segundos se envia la trampa de baja presion
            Agentuino.TrampaPresionBaja(RemoteIP, ip, locUpTime);
            tiempoPresionBaja = millis();
        }
    } else {
        tiempoPresionBaja = 0;
        delay(50);
        locUpTime = locUpTime + 5;
    }
    if (EstadoPresionAlta == HIGH) {
        if ( millis() - tiempoPresionAlta >= 5000 ) {     // Cada 5
        //segundos se envia la trampa de alta presion
            Agentuino.TrampaPresionAlta(RemoteIP, ip, locUpTime);
            tiempoPresionAlta = millis();
        }
    } else {
        tiempoPresionAlta = 0;
        delay(50);
        locUpTime = locUpTime + 5;
    }
    if (EstadoMagnetico != UltimoEstadoMagnetico){
```

```
                if (EstadoMagnetico == LOW) {
                        Agentuino.TrampaPuertaCerrada(RemoteIP,ip,locUpTime);
                        delay(50);
                        locUpTime = locUpTime + 5;
                } else {
                        Agentuino.TrampaPuertaAbierta(RemoteIP,ip,locUpTime);
                        delay(50);
                        locUpTime = locUpTime + 5;
                }
                UltimoEstadoMagnetico = EstadoMagnetico;
        }
        Agentuino.listen();     // Se escuchan solicitudes SNMP
        locUpTime = millis()/10;
}
```

La función loop verifica el estado de los puertos donde se encuentran los sensores y en caso de existir un evento (nivel lógico) disparador de trampa esta se envía mediante la función trampa correspondiente a tal evento. Por ejemplo, en el código se observa que de existir un nivel lógico alto en el puerto del sensor de humo se envía la trampa TrampaHumo que notifica que existe detección de humo y se reenvía cada 5 segundos mientras exista detección de humo. Todas las trampas se envían cada 5 segundos, a excepción de las trampas de apertura/cierre de puerte que se envían una única vez al abrirse cerrarse esta. Todos los puertos con sensores que generen mensajes de trampa son verificados para luego escuchar solicitudes SNMP mediante la función Agentuino.listen(), la cual responde apropiadamente de existir alguna solicitud; finalmente se actualiza la variable locUpTime que almacena el tiempo de encendido del prototipo y que debe ser en decenas de milisegundos.

- **Actividad J: Realizar pruebas del Arduino, sensores y placa Ethernet y comprobar un correcto monitoreo de los sensores**

  En esta actividad se hizo la verificación del funcionamiento de los sensores con la comunicación SNMP, comprobando que se obtuviesen las respuestas adecuadas por parte del prototipo ante distintas solicitudes SNMP y eventos disparadores de trampas. La forma más básica de comprobar el funcionamiento de la comunicación SNMP es mediante el uso de la herramienta

Net-SNMP (la cual puede descargarse de forma gratuita de internet), probada en este caso caso en un sistema operativo Windows; esta herramienta permite generar y recibir mensajes SNMP desde la consola del computador en distintas versiones del protocolo, aunque para este caso se usó para verificar únicamente la respuesta a mensajes tipo *get*, *set* y *getnext*.

Al realizarse las pruebas el prototipo tenía la dirección IP asignada 192.170.67.8, por lo que los comandos están referidos a esta dirección IP. Comenzando con un mensaje de respuesta tipo *get*, se emitió un comando en cónsola para acceder a la información de contacto, temperatura medida y lectura del puerto analógico 3 del Arduino. Al realizarse, se obtuvo un resultado como el que se observa en la imagen 3.18, en la cual se puede observar una respuesta satisfactoria de las variables solicitadas. De manera similar a esta, se pueden acceder a las demás OIDs típicas del grupo System como sysDescr (que contiene una descripción del agente) y sysLocation (que contiene su ubicación) o cualquier otra OID configurada en el Arduino para ser respondida.

El otro tipo de mensajes a los que un agente SNMP debe responder es a un

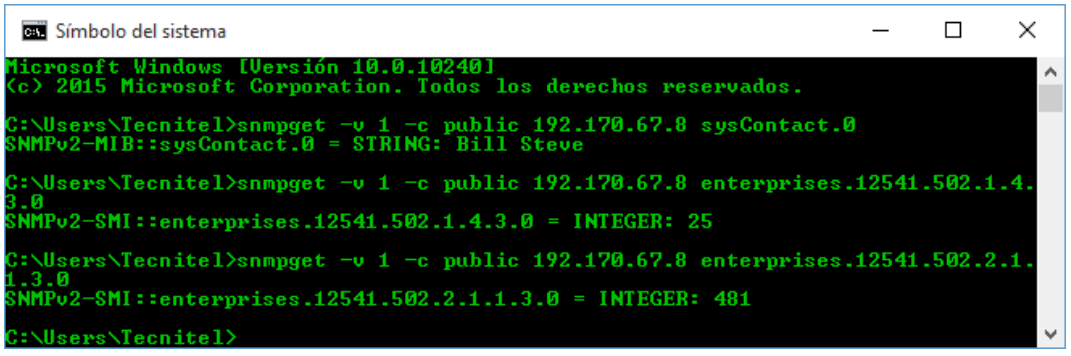


Figura 3.18: Respuesta del prototipo a mensajes tipo *get*

mensaje tipo *getnext*, el cual se puede invocar con el comando snmpwalk en consola. Este comando solicita la información de todas las OIDs que ramifican de una OID especificada. El prototipo fue configurado para que respondiera sólo a la información contenida en el grupo System del árbol MIB y en el grupo 1.3.6.1.4.1.12541.502 que es el OID propio del prototipo y del cual todas

las OIDs resultan ramas. El resultado de invocar el comando solicitando información al prototipo mostró como resultado una imagen como la 3.19, en la cual se puede observar que el prototipo responde correctamente ante este tipo de solicitudes. De manera similar el prototipo es capaz de responder directamente al comando snmpgetnext de la cónsola y entregar la información de la siguiente OID a la indicada.

A pesar de que el prototipo final no lleva ningún elemento al que se le va-

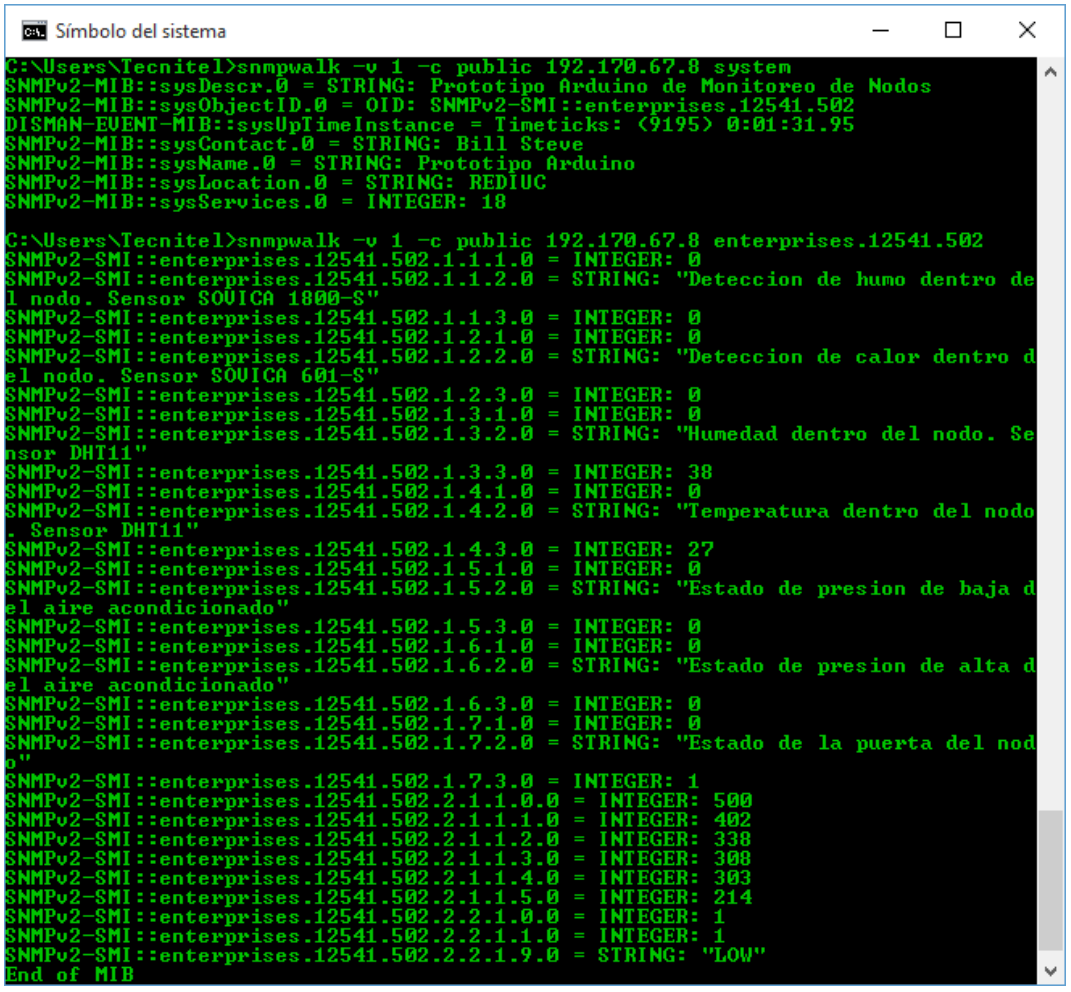
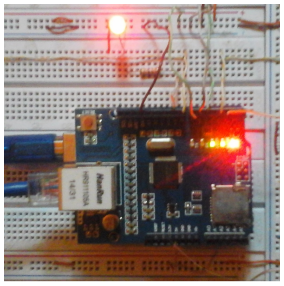
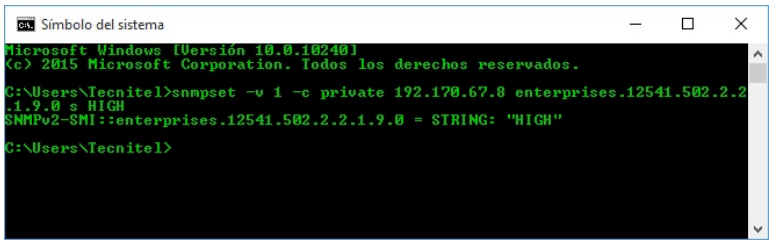


Figura 3.19: Respuesta del prototipo a mensajes tipo *getnext*

ya a asignar un valor o fijarle un estado, se dejó planteada esta posibilidad mediante el uso de mensajes *set* en uno de sus puertos, con el cual se podría



(a) LED encendido

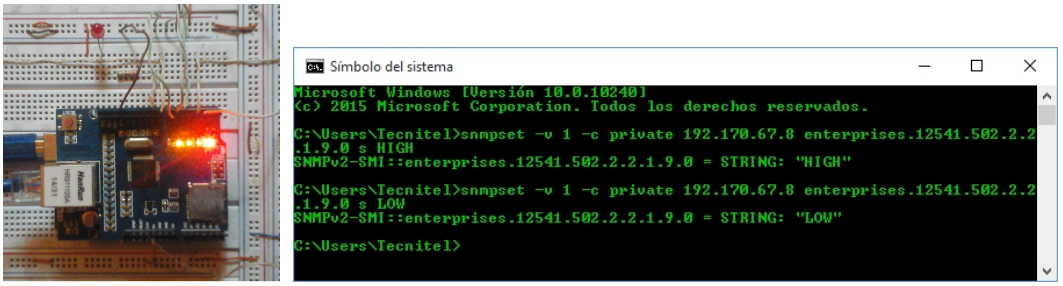


(b) Comandos *set* y *get* del estado del LED

Figura 3.20: LED encendido mediante un comando *set* de cónsola

encender o apagar un equipo o habilitar algún elemento o acción dentro del nodo. Para comprobar el correcto funcionamiento ante mensajes del tipo *set* se hicieron nuevamente pruebas mediante la cónsola del computador y se hizo la comprobación conectando un LED a un pin establecido para cambiar su estado a un nivel lógico alto (al recibir un mensaje *set* con el string "HIGH") o un nivel lógico bajo (al recibir un mensaje *set* con el string "LOW"), con lo cual se puede apagar o encender el LED según se desee. Al emitir el comando snmpset en la cónsola de Windows con el string "HIGH" el LED se enciende (figura 3.20a) y se puede observar que se sobreescribe el valor asociado en la OID objetivo y que al acceder nuevamente al valor con un comando *get* el prototipo responde con el valor "HIGH" (figura 3.20b). Cuando se emite el comando snmpset en la cónsola de Windows con el string "LOW" el LED se apaga (figura 3.21a) y se puede observar que al acceder nuevamente al valor con un comando *get* el prototipo responde con el valor "LOW" (figura 3.21b).

Por último, para verificar el funcionamiento de las trampas se hizo uso del programa SNMP Trap Watcher, el cual captura únicamente trampas por el puerto UDP 162 pero resulta muy sencillo de utilizar y configurar. Al generarse los eventos disparadores de trampas estas se generan correctamente y pueden capturarse con el programa, mostrándose una imagen similar a la 3.22. Al expandir el contenido de una de las trampas recibidas se puede observar que se recibe correctamente la información que contiene, tal como se observa en la figura 3.23.



(a) LED apagado **(b)** Comandos *set* y *get* del estado del LED

Figura 3.21: LED apagado mediante un comando *set* de cónsola

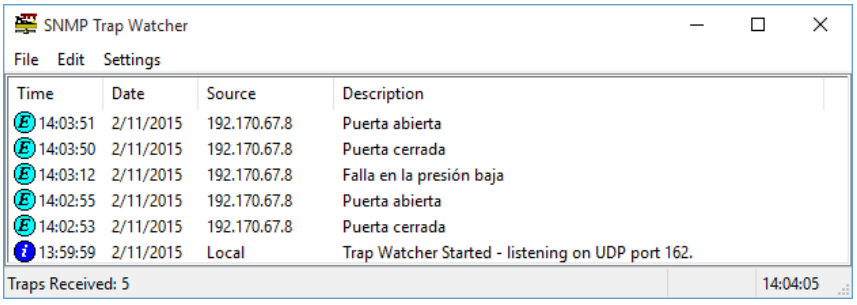


Figura 3.22: Captura de trampas utilizando SNMP Trap Watcher

3.1.5. Fase 5: Implementación

- **Actividad K: Instalar el prototipo en el nodo y realizar pruebas**
  Con el código desarrollado del prototipo y este último funcionando correctamente con los sensores, se procedió a realizar un montaje de los elementos del prototipo en baquelita de manera que los sensores que los conforman se pudiesen acoplar posteriormente. El montaje fue protegido con una caja protectora e instalado en el nodo, tal como se puede observar en la imagen 3.24. Con este montaje se realizaron las pruebas dentro del nodo para verificar el funcionamiento y correcto acoplamiento del prototipo a los sensores. Se hizo además el montaje de los sensores dentro del nodo y el paso de cables hasta el prototipo; al final se tuvo que prescindir del presostato de baja presión debido a que el que se disponía no funcionaba en el nivel de presión necesario,

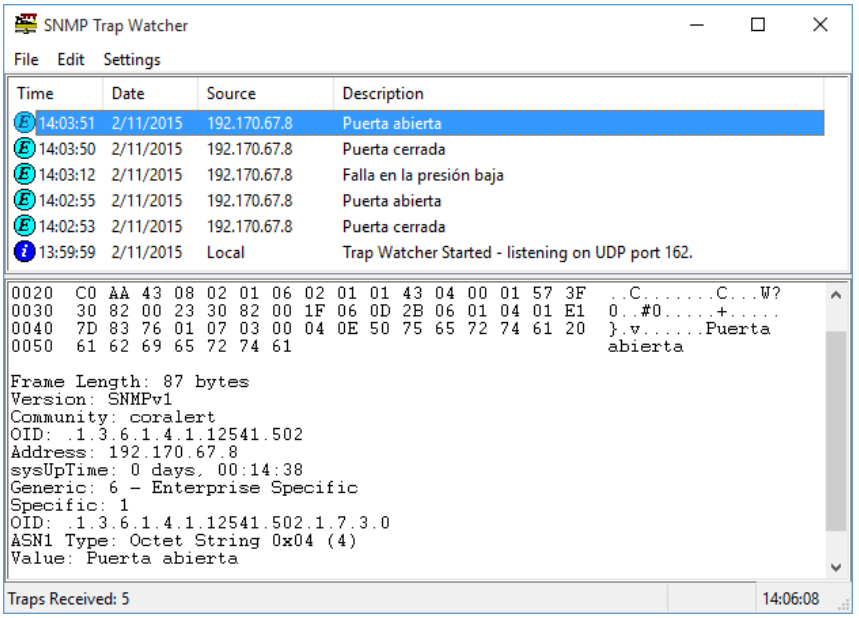


Figura 3.23: Información contenida en una trampa de puerta abierta vista en SNMP Trap Watcher

por lo que no se conectó al prototipo.

Para energizar el prototipo se hizo uso de una fuente externa de 12 voltios, lo que generó fallas en la tarjeta ethernet del Arduino, la cual se apagaba y encendía de manera impredecible. Según se averiguó, y constatando con el prototipo, este nivel de alimentación a pesar de ser apto para la tarjeta **Arduino UNO R3** genera un exceso de calor disipado en su etapa de regulación lineal. Se optó por usar entonces una alimentación de 5 voltios mediante el puerto USB del Arduino para evitar la etapa de regulación y asegurar el correcto funcionamiento del prototipo, conectando a su vez el transformador de 5 voltios a un UPS y asegurar una fuente constante de energía. Luego de energizar el prototipo se procedió a comprobar el funcionamiento del prototipo conectando este directamente a un computador, con el que se generaron solicitudes SNMP y que hizo de servidor receptor de trampas; con esto se pudo verificar la correcta generación de trampas con los sensores y funcionamiento del prototipo en el protocolo SNMP.

- **Actividad L: Realizar montaje final del prototipo**

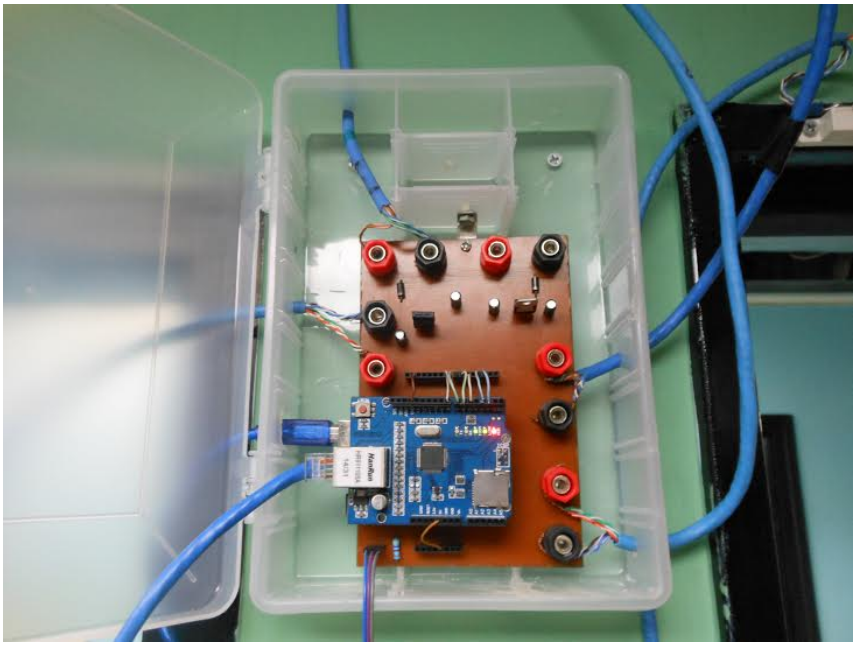


Figura 3.24: Montaje del prototipo de monitoreo en el nodo de comunicaciones

El prototipo de monitoreo se incorporó a la red universitaria REDIUC, ajustándose para esto con una dirección IP dentro la red, las comunidades SNMP adecuadas y la dirección IP de destino del servidor de trampas SNMP, para posteriormente acoplarse a los softwares de monitoreo de REDIUC. Se comprobó que el equipo pudiese responder a los pings dentro de la red y a la solicitudes SNMP, así como que en el servidor de destino de trampas SNMP se recibiesen estas; las solicitudes SNMP dieron resultados similares a los mostrados en la imágenes 3.18, 3.20b y 3.19, mientras que las trampas fueron confirmadas de recibirse en el servidor de monitoreo por la administración de REDIUC.

Para validar la calidad de las lecturas de temperatura hechas por el prototipo se hizo uso de una pistola láser medidora de temperatura para tener una medida de comparación. En un momento dado que el prototipo indicaba una temperatura de 25°C, las lecturas mediante la pistola láser indicaban una medición de 23,7°C (figura 3.25), lo cual es acorde al grado de error del sensor DHT11 [17]. Las calidad de las lecturas de humedad no pudieron verificarse ya que no se disponía de otro sensor medidor de humedad relativa.

Figura 3.25: Temperatura medida por la pistola láser

Para finalizar la implementación del prototipo se desarrolló un archivo MIB que permitiese referirse a las OID del prototipo de un modo más sencillo. El prototipo fue dejado en funcionamiento en el nodo de comunicaciones de la Facultad de Ingeniería de la Universidad de Carabobo.

- **Actividad M: Realizar el manual del usuario del prototipo**
  Con el prototipo implementado y funcional en el nodo se procedió a desarrollar un manual que resumiese la información de este proyecto a manera de dejar a los administradores del prototipo la información necesaria para acceder a las capacidades de este, así como replicarlo, detectar/corregir errores de funcionamiento y expandir capacidades. El manual fue entregado a los administradores del prototipo junto con un CD que contiene los archivos involucrados en su funcionamiento, incluyendo librerías necesarias para su funcionamiento y los códigos desarrollados en este trabajo especial de grado.

Capítulo IV

Análisis, interpretación y presentación de los resultados

4.1. Resumen de Características como Agente SNMP

El prototipo producto de esta investigación es un agente SNMP con una serie de OIDs definidas para funciones específicas. Las OIDs que utiliza el prototipo fueron desarrolladas según un esquema de árbol recomendado, con un grupo de tres OIDs por cada sensor, las cuales contienen un índice (para identificación de existir varios sensores), una descripción y la magnitud de medición. La magnitud de la medición es un valor entero que indica directamente las lecturas continuas como lo son la temperatura y la humedad, o que resulta en un 0 o un 1 en el caso de una variable discreta (siendo esta 1 para detección de humo, calor, falla de presión o detección de apertura de puerta). La tabla que se muestra en la figura 4.1 resume las OIDs de mediciones que el prototipo es capaz de responder. Como se observa no todos los puertos del prototipo se utilizan, esto debido a que deben reservarse para el correcto funcionamiento de la placa **Arduino Ethernet Shield**. Los puertos utilizables que no poseen un sensor se encuentran incorporados en el código principal del prototipo, por lo que están preparados para medir un sensor que se agregue posteriormente, aunque no poseen ni índice ni descripción. El modo en que funcionan

los puertos es distinto según si es analógico o digital y según el tipo de mensaje SNMP que soporte; esto se encuentra detallado en el manual de implementación para facilitar a los usuarios el agregado de sensores. En la figura 4.2 se encuentra la estructura MIB que posee el prototipo.

Pin	Sensor conectado	GET	SET	TRAP	OID	TRAP OID
A0	Ninguno	✓			1.3.6.1.4.1.12541.502.2.1.1.0.0	
A1	Ninguno	✓			1.3.6.1.4.1.12541.502.2.1.1.1.0	
A2	Ninguno	✓			1.3.6.1.4.1.12541.502.2.1.1.2.0	
A3	Ninguno	✓			1.3.6.1.4.1.12541.502.2.1.1.3.0	
A4	Ninguno	✓			1.3.6.1.4.1.12541.502.2.1.1.4.0	
A5	Ninguno	✓			1.3.6.1.4.1.12541.502.2.1.1.5.0	
D0	Ninguno	✓			1.3.6.1.4.1.12541.502.2.2.1.0.0	
D1	Ninguno	✓			1.3.6.1.4.1.12541.502.2.2.1.1.0	
D2	Sensor de humo	✓		✓	1.3.6.1.4.1.12541.502.1.1.1.0 1.3.6.1.4.1.12541.502.1.1.2.0 1.3.6.1.4.1.12541.502.1.1.3.0	1.3.6.1.4.1.12541.502.1.1.3.0
D3	Sensor de calor	✓		✓	1.3.6.1.4.1.12541.502.1.2.1.0 1.3.6.1.4.1.12541.502.1.2.2.0 1.3.6.1.4.1.12541.502.1.2.3.0	1.3.6.1.4.1.12541.502.1.2.3.0
D5	Sensor de temperatura y humedad	✓			1.3.6.1.4.1.12541.502.1.3.1.0 1.3.6.1.4.1.12541.502.1.3.2.0 1.3.6.1.4.1.12541.502.1.3.3.0 1.3.6.1.4.1.12541.502.1.4.1.0 1.3.6.1.4.1.12541.502.1.4.2.0 1.3.6.1.4.1.12541.502.1.4.3.0	
D6	Presostato de baja	✓		✓	1.3.6.1.4.1.12541.502.1.5.1.0 1.3.6.1.4.1.12541.502.1.5.2.0 1.3.6.1.4.1.12541.502.1.5.3.0	1.3.6.1.4.1.12541.502.1.5.3.0
D7	Presostato de alta	✓		✓	1.3.6.1.4.1.12541.502.1.6.1.0 1.3.6.1.4.1.12541.502.1.6.2.0 1.3.6.1.4.1.12541.502.1.6.3.0	1.3.6.1.4.1.12541.502.1.6.3.0
D8	Switch magnético	✓		✓	1.3.6.1.4.1.12541.502.1.7.1.0 1.3.6.1.4.1.12541.502.1.7.2.0 1.3.6.1.4.1.12541.502.1.7.3.0	1.3.6.1.4.1.12541.502.1.7.3.0
D9	Ninguno	✓	✓		1.3.6.1.4.1.12541.502.2.2.1.9.0	

Figura 4.1: Tabla resumen de las características de agente SNMP

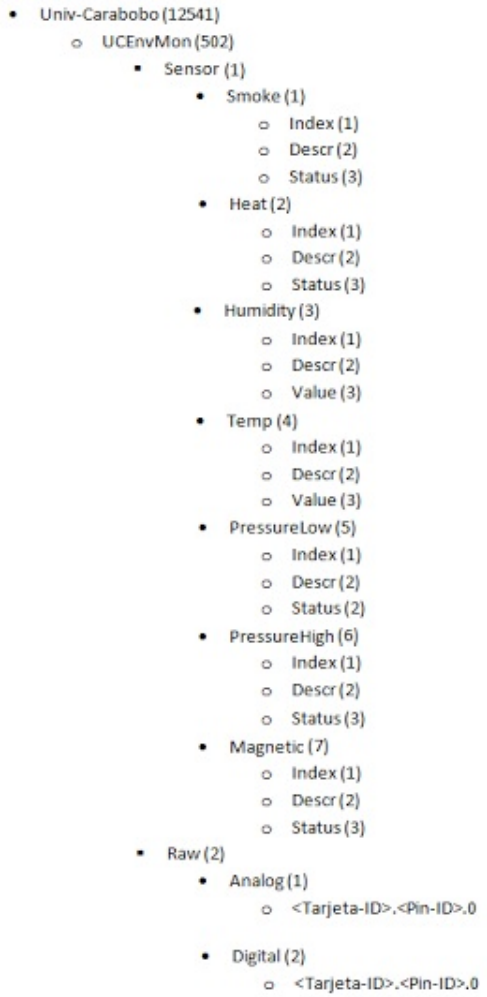


Figura 4.2: Resumen de estructura MIB del prototipo

4.2. Evaluación del Prototipo

4.2.1. Compatibilidad

Para verificar la capacidad del prototipo de funcionar como un agente SNMP en una red de monitoreo hace falta verificar su capacidad de acoplarse a los sistemas operativos y herramientas involucradas en el monitoreo SNMP de una red, es por eso que el prototipo fue probado con distintos softwares, diferentes a los ya utilizados, en los sistemas operativos Windows (para ordenadores) y Android (para dispositivos móviles).

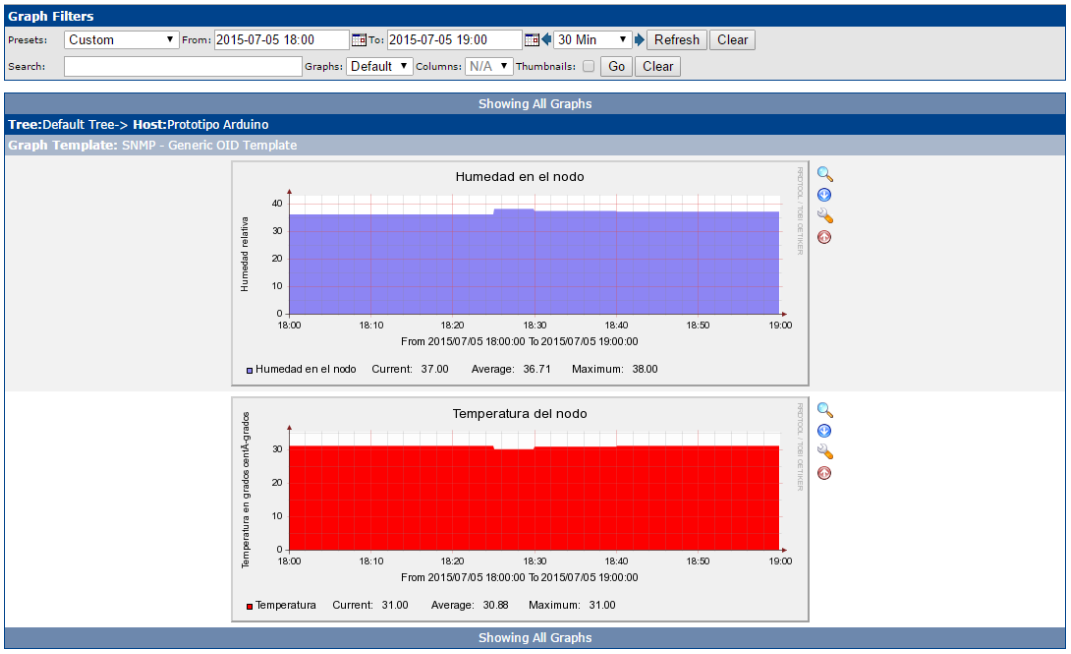


Figura 4.3: Gráficas de las variables del prototipo generadas en Cacti

El prototipo fue desarrollado con la finalidad de agregarlo a la red de REDIUC, la cual actualmente usa la herramienta computacional Cacti para monitorear las condiciones de su red y graficarlas, es por este motivo que es necesario que el prototipo se pueda adaptar correctamente a esta herramienta. La herramienta Cacti genera solicitudes *get* de manera periódica para graficar variables, es decir que puede utilizarse para graficar la temperatura y la humedad relativa medida por el prototipo. Se procedió entonces a configurar Cacti con el prototipo como un nuevo dispositivo y se agregaron las OIDs de las variables de temperatura y humedad, con lo efectivamente se pudieron graficar las variables solicitadas, tal como se puede observar en la figura 4.3 en la que se muestran la gráficas de la temperatura y humedad medidas por el prototipo.

Para verificar la capacidad del prototipo de comunicarse con el sistema operativo Android se hizo uso de la aplicación SNMP Management System para plataforma Android, la cual permite que el dispositivo donde se ejecuta se comporte como un

Figura 4.4: Respuesta del prototipo en SNMP Management System a la solicitud *getnext* del grupo System

administrador SNMP y pueda hacer solicitudes *get*, *getnext* y *set*, además de generar y recibir trampas si se desea. Usando esta aplicación se puede hacer, de manera similar a como se hizo en cónsola, una petición a las OID del grupo System a las cuales el prototipo responde correctamente, como se puede ver en la imagen 4.4. Con el uso de esta aplicación también se pueden realizar solicitudes del tipo *get* y *set* correctamente al prototipo, así como recibir las trampas que este genera.

4.2.2. Costo

El prototipo de monitoreo fue desarrollado con la finalidad de ser fácilmente replicable, de manera de poder implementarse en todos los nodos que abarca REDIUC. Para lograrlo es necesario verificar el costo de los elementos que involucra el prototipo y comprobar la rentabilidad de implementarse en uno de estos nodos. En la figura 4.5 se resume el costo de los elementos involucrados en el prototipo con precios tomados directamente de `www.mercadolibre.com.ve` y `www.amazon.com`, sin conversión monetaria de ningún tipo para el cálculo. En el apéndice A se especifican los enlaces usados para referir el precio de los productos. El costo mostrado en la imagen no incluye el costo los elementos involucrados en el montaje final

(baquelita, caja, etc.), y dado que los precios fueron obtenidos de páginas de compra/venta entre personas (en vez de empresas) los costos no incluyen ningún tipo de costo de envío o transporte ya que depende del acuerdo entre los involucrados.

Elemento	Bs.	$
SOVICA 1800-S	13.000	No disponible
SOVICA 601-S	16.800	No disponible
Presostato de baja	3.499	21,36
Presostato de alta	3.499	16,25
DHT11	2.450	2,59
Switch magnético	1.616	6,38
Arduino UNO R3	9.999	15,9
Arduino Ethernet Shield	No disponible	6,54
Total	50.863	69,02

Figura 4.5: Costo de los elementos del prototipo de monitoreo

4.2.3. Limitaciones

El prototipo desarrollado en esta investigación es capaz de funcionar efectivamente como un agente SNMP, sin embargo presenta algunos limitaciones en cuanto al modo en que funciona y que deberían tomarse en cuenta a la hora de implementarse. En cuanto al código con el que funciona se pueden mencionar las siguientes limitaciones:

- El prototipo se comunica únicamente mediante la versión 1 del protocolo SNMP, el cual actualmente ha evolucionado hasta una tercera versión que incluye nuevas medidas de seguridad para encriptar la información contenida en las tramas e incluir autenticación de usuario. Ya que la primera versión de SNMP no incorpora estas medidas de seguridad se considera en comparación un protocolo inseguro.
- La librería Agentuino utilizada para desarrollar el prototipo soporta únicamente que en un mensaje del tipo *get* se solicite la información de una sola

OID, por lo que el prototipo es incapaz de responder a un mensaje que solicite simultáneamente la información de varias OIDs.

- El prototipo posee una variable llamada locUpTime que almacena el tiempo que lleva este encendido, cuando se hace una solicitud a la OID sysUpTime del grupo System se responde con el valor almacenado en esta variable; la variable se actualiza con la ayuda del comando *millis()* en el código Arduino, la cual al invocarse genera que el Arduino responda con el tiempo de encendio en milisegundos, almacenado en una variable unsigned long de 4 bytes que se desborda después de aproximadamente 50 días. Esto significa que el valor de la OID sysUpTime se vuelve erróneo después de 50 días de encendido del prototipo, pues este va a reiniciarse desde cero. Esto no afecta el funcionamiento del prototipo, pero resulta en un valor erróneo al leerse el tiempo de encendido del prototipo (OID sysUpTime) y en la hora contenida en las trampas.

La capacidad del prototipo de realizar mediciones también se encuentra limitado por los sensores que lo conforman, es decir que lo limitan las capacidades que tengan estos y que fueron mencionados previamente en la actividad de selección y adquisición de sensores de la etapa 3.1.2. Las limitaciones del Arduino también deben considerarse, ya que este posee un límite de la corriente o energía capaz de entregar para energizar sensores activos o placas que se quieran acoplar, además de poseer un número limitado de puertos disponibles para ser utilizados. Estas son consideraciones que deben tomarse en cuenta en caso de querer expandir las capacidades del prototipo o de modificar alguna de sus características. En el prototipo desarrollado todos los puertos son utilizados o están reservados para la conexión futura de sensores o porque deben estar libres para que la **Arduino Ethernet Shield** funcione correctamente. Las limitaciones expuestas en esta sección se mencionan de un modo más profundo en el manual de implementación del prototipo junto con las recomendaciones para abarcarlas.

Capítulo V

Conclusiones y Recomendaciones

5.1. Cumplimiento de los Objetivos de la Investigación

Esta investigación surgió en respuesta a una necesidad de la Universidad de Carabobo de monitorear las condiciones de los nodos que conforman su infraestructura de red y preservar de esta manera la calidad del servicio que ofrece. Tal como se planteó, esta investigación logró desarrollar de manera efectiva un prototipo actual de monitoreo de variables capaz de comunicarse con distintas plataformas y softwares computacionales, volviéndolo más flexible a las herramientas de los administradores de la red y siendo fácilmente replicable a un bajo costo. El prototipo fue implementado en REDIUC y acoplado a las herramientas computacionales que en ella se utilizan (Cacti), por lo que el prototipo, los códigos involucrados en su funcionamiento y el manual de implementación quedaron a disposición de la Universidad de Carabobo. El manual de implementación desarrollado en esta investigación resume las características de interés a cualquiera interesado en comprender rápidamente el prototipo y facilitar el proceso de réplica, modificación, expansión y corrección de posibles errores, incluyendo algunas recomendaciones que bien se pueden tomar en cuenta al realizar estos procesos.

5.2. Recomendaciones

El prototipo desarrollado a pesar de cumplir con su función de agente SNMP presenta una serie de limitaciones expuestas en la sección 4.2.3. Estas limitaciones a su vez son el punto de partida para nuevas investigaciones que intenten desarrollar un sistema de monitoreo basado en hardware libre. La investigación desarrollada puede servir como un precedente para el desarrollo de futuras investigaciones y mejoras que incluyan:

- Desarrollar las librerías para Arduino que permitan responder a solicitudes SNMP en sus versiones 2 y 3. Estas librerías se desarrollarían con la finalidad de que el prototipo funcione con protocolos más seguros y se adapte a nuevos programas que los utilicen.
- Desarrollar las librerías para Arduino que permitan que se comunique mediante otros protocolos de monitoreo de redes como CMIP y NETCONF. Ya que SNMP no es el único protocolo de administración de redes se podrían desarrollar las librerías adecuadas para que se comunique en otros protocolos con características y beneficios propios, permitendo al prototipo funcionar en un abanico nuevo de programas.
- Desarrollar un prototipo de monitoreo de nodos capaz de comunicarse mediante IPv6. El prototipo producto de esta investigación hace uso de la **Arduino Ethernet Shield** para obtener comunicación de red; este módulo se basa en el chip Wiznet W5100 que funciona únicamente en redes IPv4, por lo que para obtener comunicación IPv6 es necesario sustituir el módulo de red. Existen módulos de Arduino basados en el chip ENC28J60 que permiten la comunicación mediante IPv6 y resultan más económicas que el módulo de red utilizado en el prototipo, sin embargo requieren el uso de librerías distintas y sería necesario modificar las librería del prototipo que involucren comunicación de red.

- Hacer uso de la funcionalidad de tarjeta micro SD del **Arduino Ethernet Shield** para guardar un historial de la comunicación SNMP y datos de mediciones para poder graficar con otros softwares.

Apéndice A

Enlaces utilizados para Cálculos de Costo del Prototipo

Las páginas web de estos productos fueron consultadas en 11 de noviembre de 2015:

SOVICA 1800-S:

- http://articulo.mercadolibre.com.ve/MLV-447817686-detector-ionico-1800-s-plus-sovica-electronics-_JM

SOVICA 601-S:

- http://articulo.mercadolibre.com.ve/MLV-446486197-detector-termico-sovica-601s-601-_JM

Presostato de baja presión:

- http://articulo.mercadolibre.com.ve/MLV-449018102-presostato-de-motores-de-refrigeracion-prst-lw-25-a-80-psi-_JM

- http://www.amazon.com/Low-Pressure-Switch-Open-PSI/dp/B007IBV6N4/ref=sr_1_2?s=hi&ie=UTF8&qid=1447795250&sr=1-2&keywords=75psi+pressure+switch

Presostato de alta presión:

- http://articulo.mercadolibre.com.ve/MLV-449018119-presostato-de-motor-de-refrigeracion-prst-hh-350-250-psi-_JM
- http://www.amazon.com/Sealed-Company-SHP350250-Pressure-Switch/dp/B009PARTRI/ref=sr_1_3?ie=UTF8&qid=1446849829&sr=8-3&keywords=high+pressure+switch+350+psi

DHT11:

- http://articulo.mercadolibre.com.ve/MLV-449024744-sensor-de-humedad-y-temperatura-dht11-arduino-pic-_JM
- http://www.amazon.com/Sensitivity-Control-Temperature-Humidity-20-90%25RH/dp/B00BXWUWRA/ref=sr_1_4?ie=UTF8&qid=1444927149&sr=8-4&keywords=dht11

Switch magnético de puerta:

- http://articulo.mercadolibre.com.ve/MLV-447616328-contacto-magnetico-sm205-w-_JM
- http://www.amazon.com/Directed-Electronics-8601-Magnetic-Switch/dp/B0009SUF08/ref=sr_1_1?ie=UTF8&qid=1444927360&sr=8-1&keywords=magnetic+switch

Arduino UNO R3:

- http://articulo.mercadolibre.com.ve/MLV-449180570-arduino-uno-r3-solo-quedan-estos-_JM

- http://www.amazon.com/Arduino-A000066-Uno-Rev-3/dp/B008GRTSV6/ref=sr_1_4?s=electronics&ie=UTF8&qid=1444928070&sr=1-4&keywords=arduino+uno+r3

Arduino Ethernet Shield:

- http://www.amazon.com/Desloo-Ethernet-Micro-sd-Arduino-Duemilanove/dp/B00GIDHZHE/ref=sr_1_13?s=electronics&ie=UTF8&qid=1444927727&sr=1-

Referencias Bibliográficas

[1] León S. y Loaiza E. Diseño e implementación de un prototipo de adquisición de datos de magnitudes físicas en los nodos de REDIUC, 2002.

[2] Castro A. Sistema de control de temperatura a través de Arduino y tecnología GPRS/GSM, 2013.

[3] Sánchez E. Diseño de un sistema de control domótico basado en la plataforma Arduino, 2012.

[4] Sinclair I. *Sensors and Transducers*. Newnes, 2001.

[5] Kourosh K. y Benjamin F. *Nanotechnology-Enabled Sensors*. Springer, 2008.

[6] Arduino, Abril 2015. URL http://arduino.cc/.

[7] Evans B. *Arduino Programming Notebook*. 2007.

[8] O. Jonas y W. Samson Olsson T., Gaetano D. *Open Softwear: Fashionable prototyping and wearable computing using the Arduino*. 2008.

[9] Marshall DenHartog. *SNMP Tutorial: The Fast Track Introduction to SNMP Alarm Monitoring*, 2010.

[10] Mauro D. y Schmidt K. *Essential SNMP*. 2001.

[11] SNMP Tutorial, Noviembre 2015. URL https://www.manageengine.com/network-monitoring/what-is-snmp.html.

[12] SolarWinds Worldwide. *A Guide to Understanding SNMP*, 2013.

[13] Sovica Electronics C.A. *Detector Iónico, Modelo: 1800-S*, 2006.

[14] Atmel. *Atmel 8-bit Microcontroller with 4/8/16/32KBytes In-System Programmable Flash*, 2012.

[15] Sovica Electronics C.A. *Detector Térmico, Modelo: 601-S*, 2009.

[16] HVAC Pressure Control Switches, Noviembre 2015. URL http://www.hvacspecialists.info/control-devices/hvac-pressure-control-switches.html.

[17] D-Robotics UK. *DHT11 Humidity & Temperature Sensor*, 2010.

[18] Nasser A. Snmp traps (simple network management protocol), November 2000.