

Curso de Capacitación
Arduino y Raspberry Pi Como Herramientas Didácticas
Apunte de Profundización

Universidad Tecnológica de Chile – INACAP
Relator: Antonio D. Vásquez Briones

29 de diciembre de 2017

Índice general

1. Introducción	1
1.1. Micrонтroladores y Microprocesadores	1
1.2. Placa de Desarrollo Arduino UNO	2
1.3. Single Board Computer Raspberry Pi	3
2. Ecosistema Arduino y Control Digital	5
2.1. ¿Qué es Arduino?	5
2.2. Primeros Pasos	6
2.2.1. Obtención del Entorno Integrado de Desarrollo	6
2.2.2. Conexión de Arduino al PC y Configuración del IDE	7
2.2.3. Estructura de un Código Arduino	9
2.3. Control de Salidas Digitales	9
2.3.1. Actividad	10
2.4. Control de entradas Digitales	11
2.4.1. Actividad	13
2.5. Control de Salidas PWM	14
2.5.1. Actividad	15
2.6. Lectura de Entradas Analógicas	15
2.6.1. Actividad	16
2.7. Comunicación Serial	16
2.7.1. Actividad: Lectura de variable analógica desde el Puerto Serie	18
2.7.2. Actividad: Envío de instrucciones a Arduino a través del Puerto Serie	19
3. Lectura de Sensores y Control de Actuadores con Arduino.	20
3.1. Sensor de luminosidad LDR	20
3.1.1. Actividad	21
3.2. Sensor de Temperatura LM35	22
3.2.1. Actividad	22
3.2.2. Serial Plotter	23
3.3. Sensor Ultrasónico Parallax PING	23
3.3.1. Actividad	25
3.4. Pantalla LCD 16x2	25
3.4.1. Instalación de Librerías	27
3.4.2. Programación de Pantalla LCD	28
3.4.3. Actividad	29
3.5. Control de Servomotores	30
3.5.1. Actividad	31
3.6. Motores DC	32
3.6.1. Actividad	33
3.7. Cargas Fuertes y Corriente Alterna por medio de Relé	33

3.7.1. Actividad	34
4. Introducción a los Sistemas de Control Automático	36
4.1. Automatización	36
4.2. Control de Eventos Discretos	37
4.3. Control Regulatorio	37
4.4. Estructura general de control	38
4.4.1. Bloques del Sistema	38
4.4.2. Señales del Sistema	38
4.5. Aplicación de Arduino como Sistema de Control Regulatorio	39
4.5.1. Control Proporcional Integral y Derivativo PID	40
4.5.2. Código que Implementa el Control del Sistema	40
4.5.3. Actividad	41
5. Internet de las Cosas con Raspberry Pi	43
5.1. Primer Encendido y Configuración Para Uso Remoto	43
5.1.1. Actividad: Configuración de Raspberry para conexión remota	44
5.2. Control de GPIO's en Python	46
5.2.1. Actividad: Manejo de GPIO's desde Python	47
5.3. Servicio IoT Ubidots.com	49
5.3.1. Actividad: Cuenta y Dispositivos en Ubidots	49
5.3.2. Actividad: Envío de datos a Ubidots	51
5.3.3. Actividad: Accionamiento de dispositivos desde Ubidots	52

Resumen

El presente documento, presenta diversos apuntes y material de apoyo para el curso *Arduino y Raspberry Pi Como Herramientas Didácticas* orientado a académicos del **Área Informática y Telecomunicaciones**, dictado durante el mes de enero de 2018 en Sede Santiago Sur.

Los tópicos tratados en este curso, están orientados a profesionales de las áreas mencionadas que no cuentan con conocimientos en estas plataformas, pero si cuentan con un trasfondo sólido de conocimientos en programación estructurada y sistemas operativos, como por ejemplo, el Lenguaje C y manejo básico de Sistemas Operativos basados en Linux.

El apunte, está orientado para ser seguido junto con el avance del curso, el cual se estructura en tres capítulos que corresponden a las tres jornadas en que se dicta, de las cuales, se hace un esfuerzo para que cada una de estas, se subdivida en 4 bloques de dos horas, lo que corresponderían a las secciones de los capítulos correspondientes.

Capítulo 1

Introducción

En este capítulo introductorio, se entrega información acerca de conceptos fundamentales involucrados en dos plataformas y proyectos *Open-Source* independientes entre sí; **Arduino** y **Raspberry**. Estas plataformas, actualmente pasan por un evidente periodo de auge gracias a diversos portales en internet, al respaldo que han tenido por diversas instituciones educacionales de renombre alrededor del mundo y por sobre todo por contar con una gran y creciente comunidad que genera y comparte contenidos que permiten que muchas personas, con o sin tener una base de conocimientos en la materia, pueda poner en marcha interesantes proyectos de *hágalo usted mismo* o *Do It Yourself DIY* por sus siglas en inglés, por lo tanto, no es extraño que tanto alumnos como docentes, constantemente insistan en la utilización de este tipo de plataformas, básicamente por que parece muy fácil y esta *de moda*, cuando en realidad lo que se está haciendo es un *copy-paste* desde internet, lo cual no es del todo algo malo.

Por otro lado, el relator de esta capacitación, considera que la verdadera riqueza, está en que tenemos en nuestras manos, poderosas capacidades de cálculo que nos permitirán exprimir gran parte del conocimiento teórico que se imparte, en áreas ligadas a las Tecnologías de la Información y la Electrónica, puesto que en esencia estos artefactos, no son más que una implementación particular, de lo que desde décadas se ha venido enseñando, por lo tanto cualquier análisis desde la rigurosidad del *Electrical Engineering and Computer Science* es posible y de hecho, resulta muy enriquecedor.

El llamado es, a no caer en que nuestros alumnos sean meros repetidores de proyectos que encuentran en la web, si no que estos puedan articular e integrar implementaciones, para crear más y mejores proyectos, gracias un análisis riguroso, sustentado en conocimientos sólidos en cada una de sus áreas.

1.1. Micrcontroladores y Microprocesadores

Desde los años 90, por estos lados del mundo (Chile), con el auge de la computación y posteriormente internet, el término **Micro-Procesador**, nos resulta bastante familiar. A grandes rasgos, sabemos que se trata del *cerebro* del sistema computacional, al punto que hoy en día se nos viene de inmediato a la cabeza exponentes como, *Intel Core i7*, *AMD Ryzen*, *Intel Pentium*, *AMD A9* por nombrar algunos. Por otro lado, existen por doquier, en nuestros dispositivos, como celulares, teclados, televisores, hornos micro-ondas, etc. otro tipo de dispositivos muy parecidos en arquitectura y funcionalidades; los **Micro-Controladores**, que están siempre silenciosamente, encargándose de tareas específicas, como por ejemplo, llevar el control de la potencia y el tiempo al calentar un plato en un Micro-ondas.



Figura 1.1: Principales Fabricantes actuales de Microprocesadores y Microcontroladores.

Podemos entender inicialmente, estos dos tipos de dispositivos muy parecidos y diferentes a la vez, como bifurcaciones evolutivas de la máquina computacional. Así es, existe un mismo origen, por allá por 1971 en los albores de la compañía fabricante de *chips* **Intel**, se desarrolló el primer microprocesador **Intel 4040**. El problema que resolvía era simple: Calcular, pues era parte de las calculadoras *Busicom* de la época y más tarde el microprocesador **Intel 8086** como parte de los primeros ordenadores, es decir, servían a los seres humanos en sus tareas con el manejo y procesamiento de *información* de manera rudimentaria, para ello, se valían de los principios de la electricidad y la electrónica para tomar *información* del mundo real, procesarla a un nivel abstracto en forma de señales eléctricas y finalmente devolver el resultado final en forma de información inteligible por los seres humanos. Ese mismo potencial de cálculo desarrollado, podría ser utilizado para resolver problemas con señales eléctricas que actúan sobre elementos físicos en la industria y la vida cotidiana, para lo cual, no se requiere gran parte de los periféricos que trabajan con información *humana*. En base a esta utilidad, nace el **Microcontrolador** como un sistema computacional mínimo, pero completo inserto en un sólo chip, orientado principalmente a trabajar con señales eléctricas por sobre el manejo de información. [1]

Cabe notar, que en la actualidad la diferencia entre microprocesador y microcontrolador ha dejado de ser una línea bien definida, puesto que existen sistemas como los llamados *System On Chip* SoC que utilizan microprocesadores conocidos insertos en un mismo chip junto a otros periféricos de un sistema computacional, sin ser un microcontrolador, y los *Circuitos Integrados de Aplicación Específica* ASIC's que podrían integrar tanto microprocesadores o microcontroladores en su diseño sin que sea necesario que su fabricante especifique su diseño interno. Toda esta gama de dispositivos y aplicaciones conforman el área de especialización llama **Sistemas Embobidos** o Embedded Systems.

1.2. Placa de Desarrollo Arduino UNO

Arduino Uno, es una **Tarjeta de Desarrollo** basada en el Microcontrolador de Atmel ATmega328p, precargada con un *Bootloader* que permite la programación del microcontrolador directamente desde un puerto serial UART. Contiene los periféricos asociados que permiten que el microcontrolador entre en funcionamiento, tal cómo se puede apreciar en la imagen 1.2, donde tenemos:

- **Cristal Oscilador:** Encargado de entregar la *señal de reloj* para que el microcontrolador pueda ejecutar instrucción tras instrucción. Funciona en base al **Efecto Piezoelectrónico**.
- **Etapa de alimentación:** Regula el voltaje aplicado como alimentación a ambos microcontroladores—principal y de comunicación— puede recibir de forma segura voltajes de entrada hasta los 12 volts y entregar un voltaje estable de 5 volts.
- **Comunicación USB-Serial UART:** Es un microcontrolador adicional, incluido en las últimas versiones de arduino¹, está programado para emular comunicación serial sobre usb, lo que permite programar el chip

¹Desde la versión R3 en adelante. La mayoría de las placas *arduino compatible*, no tienen este chip y traen el dispositivo CH340g, el cual no es programable y solo se encarga de la comunicación SERIAL-USB a UART-TTL

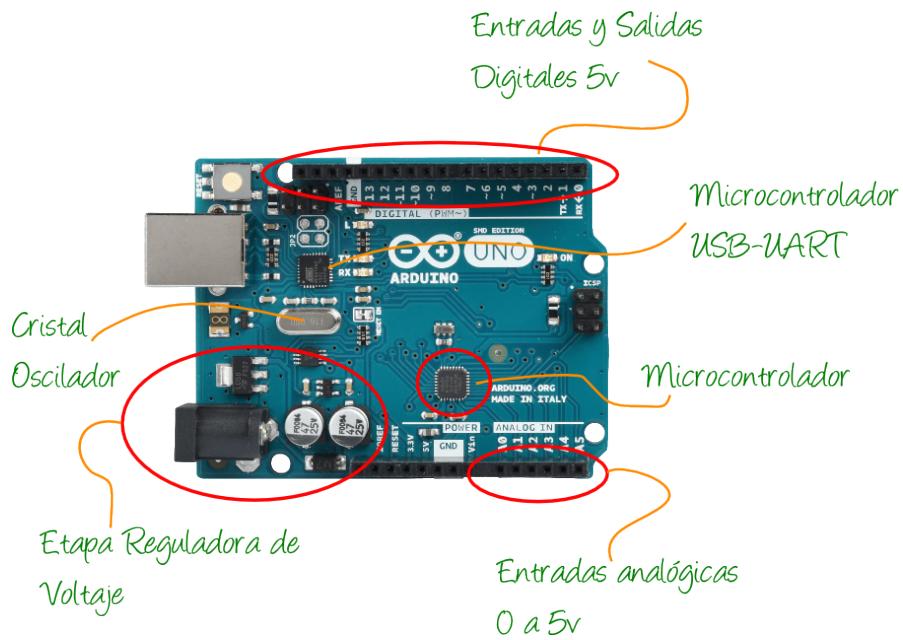


Figura 1.2: Anatomía de la Tarjeta de Desarrollo Arduino Uno.

principal y establecer comunicación entre el PC y Arduino. También es programable (DFU programming) y permite generar dispositivos usb nativos *Human Interface Device HID* como teclados, mouse, controladores midi, entre otros.

- **Pines de entradas y salidas digitales:** Son conexiones eléctricas directas con pines específicos del microcontrolador y son las encargadas de convertir señales eléctricas en bits o vice-versa al interior de un programa. Funcionan detectando o emitiendo voltajes de 0 ó 5 volts, los cuales serán interpretados a nivel de programa como estados HIGH o LOW. Habitualmente el umbral de detección se encuentra alrededor de los 3 volts, es decir, todo voltaje aplicado a un pin digital, entre 3 y 5.5 volts será considerado como estado HIGH y si esta entre 3 volts y 0 volts, será un estado LOW. [2]
- **Pines de Entradas Analógicas:** Permiten que el sistema lea en forma de voltaje, información proveniente de variables continuas por medio de diversos tipos de sensores.

1.3. Single Board Computer Raspberry Pi

Raspberry Pi, es básicamente un computador tal y como los conocemos, muy pequeño, del tamaño de una tarjeta de crédito, pero bastante reducido en recursos. Podríamos decir, que se asemeja en desempeño a un computador de hace unos ocho a diez años atrás, aunque evidentemente tiene más y mejor tecnología que aquellos de antaño y lo mejor, a un muy bajo costo.

En términos de hardware, se trata de un sistema con arquitectura ARM—como los smartphone—integrado con memoria ram y capacidades de video en un mismo chip² acompañado con periféricos como un Bus USB, un lector de tarjetas SD, interfaz de red y una interfaz física de video cómo HDMI o salida analógica de video,

²Aunque esto cae en la definición de microcontrolador, la complejidad del procesador califica a todo el sistema dentro de la categoría SoC

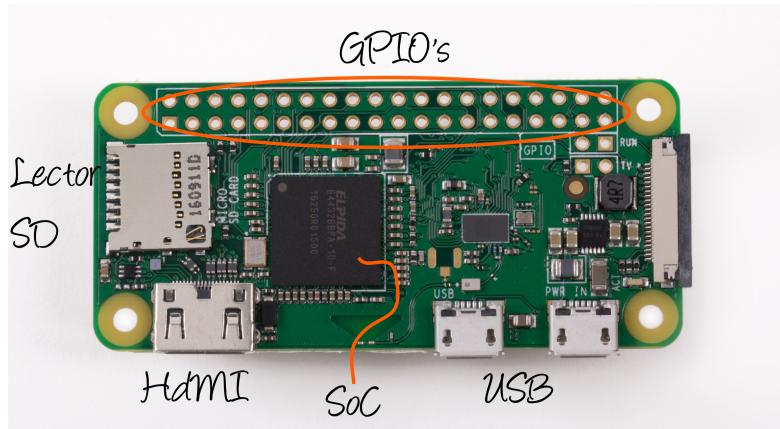


Figura 1.3: Raspberry Pi Zero (6.5 cm de longitud)

adicionalmente, posee interfaces específicas que los computadores utilizan para manejar circuitería interna en redes de bajo nivel, como buses SPI, I2C, los clásicos UART para comunicación Serial y unos pines de entrada o salidas que serían algo así como bits físicos, llamados GPIO, *General Purpose Input Output*. Todas las variantes de raspberry, corresponden a la misma arquitectura, por ejemplo en la figura 1.3, se puede ver el caso de la más pequeña de todas, Raspberry Pi Zero.

En términos de conocimientos necesarios para dominar sistemas con Raspberry Pi, cabe notar que se trata de un computador, por lo tanto, el conocimiento requerido depende más del sistema operativo que del dominio del hardware. Por excelencia, desde su creación ha sido la variante de Debian, **Raspbian**, que contiene precargados numerosos software que permiten probar funcionalidades básicas, pero no se debe olvidar que su potencial es el mismo que cualquier sistema Linux, por ende puede ser programado en diversos lenguajes, al igual que cualquier sistema operativo. Adicionalmente existen muchas *distros*³ linux, específicas para Raspberry e incluso existe Windows CE, un sistema ampliamente utilizado por computadores embebidos a nivel industrial.

³Personalizaciones de linux como Ubuntu, Debian, Linux Mint, Fedora, entre otras

Capítulo 2

Ecosistema Arduino y Control Digital

2.1. ¿Qué es Arduino?



Arduino es una plataforma de **Electrónica de Código Abierto**—*Open Source Electronics Platform*— basada en hardware y software que se caracterizan por su facilidad de uso. Las tarjetas de desarrollo Arduino pueden leer entradas desde el mundo físico real, realizar un procesamiento a nivel de datos y devolver algún tipo de interacción con el mundo real, en forma de Luz, Movimiento, Sonido, etc. Para usar un arduino, se hace programándola directamente desde un computador por medio del *Lenguaje Arduino*¹ heredado de un proyecto anterior llamado *Wiring*, el Entorno Integrado de Desarrollo (IDE por sus siglas en inglés) está basado también en un proyecto similar llamado *Processing*.²

A lo largo de los años, Arduino ha sido el *Cerebro* de miles de proyectos, desde objetos cotidianos hasta instrumentos y aplicaciones científicas. Gran parte del éxito de esta plataforma, se debe a una gran comunidad alrededor del mundo de *Makers*, estudiantes, entusiastas, artistas, programadores y profesionales que han permitido que este proyecto *open-source* siga avanzando gracias a la contribucion de increibles cantidades de conocimiento accesible a través de la web que es de gran ayuda tanto para principiantes como expertos en el área.

Arduino nace en el *Ivrea Interaction Design Institute* como una herramienta de fabricación rápida de prototipos, que permitiría que estudiantes sin un respaldo de conocimientos en electrónica y programación pudieran desarrollar proyectos tecnológicos. Tan rápidamente como se lograron alcanzar grandes comunidades el proyecto en si mismo comenzó a adaptarse a diversas necesidades de distintas áreas, al punto que hoy en día existe un amplio ecosistema tanto hardware como software que permite desarrollar desde simples aplicaciones hasta resolver desafíos actuales

¹Es una API implementada en lenguaje C++ con instrucciones intuitivas que permiten realizar tareas simples.

²Una API implementada en Java para programar contenido audiovisual.

de la industria 4.0 como el control de procesos, internet de las Cosas, Domótica, impresión 3d, maquinas CNC, entre otras áreas. [3]. En la imagen 2.1 se puede apreciar el software Processing, la Placa Wiring S, y la primera versión de Arduino *Duemilanove*.

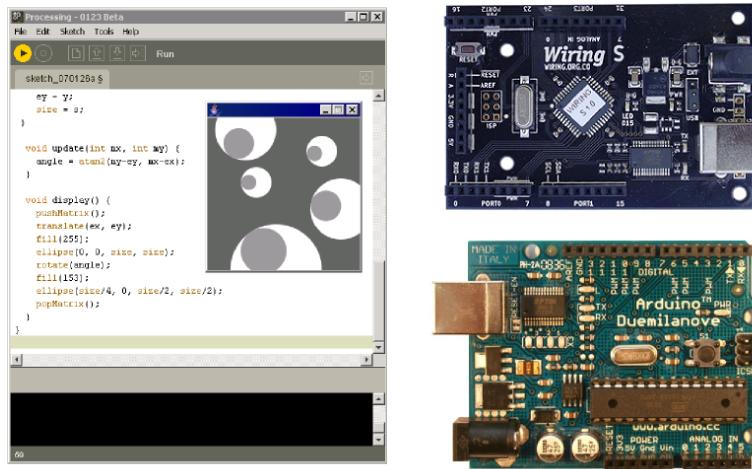


Figura 2.1: IDE processing, Placa Wiring S y Arduino Duemilanove.

2.2. Primeros Pasos

2.2.1. Obtención del Entorno Integrado de Desarrollo

El primer paso, para comenzar con Arduino, es descargar el Entorno Integrado de Desarrollo del mismo nombre, para ello es necesario un computador con acceso a internet. Habitualmente, es posible descargarlo desde el siguiente sitio.



Figura 2.2: Sitio de descarga del IDE Arduino.

En la imagen 2.2, se puede apreciar que existen dos instaladores de Windows, el primero es una instalación típica como cualquier software, mientras que el segundo *Non-admin Installer* sirve para utilizar el software en maquinas que no tengan habilitados los permisos de administrador, lo cual, resulta de gran utilidad, pues es el caso de la mayoría de los PC's institucionales.

La instalación, como ha de esperarse en Windows, es presionar el botón siguiente hasta finalizar, con ello se instalan los drivers y todo lo necesario para que el sistema funcione. En sistemas Linux como Debian basta la instrucción [sudo apt-get install arduino] para instalar el software, puesto que se encuentra incluido en los repositorios oficiales.

2.2.2. Conexión de Arduino al PC y Configuración del IDE

El siguiente paso, es conectar el Arduino al PC por medio del cable USB A-B o USB tipo D a un puerto cualquiera de nuestro computador, cabe notar que este puerto brindará energía para la mayoría de las experiencias que realizaremos. Una vez conectada, lo normal es que parpadeen algunos led's de la placa, dependiendo del programa que haya estado previamente cargado en cada una de las placas, puede que no se comporten todas igual.

Una de las maneras de asegurarnos de que toda la conexión está bien y que no hay problema de drivers en windows, es revisar el administrador de dispositivos (win+R devmgmt.msc) y buscar entre los dispositivos *Puertos COM & LPT* debería estar nuestra arduino en algún puerto COM cualquiera, tal como se muestra en la imagen 2.3

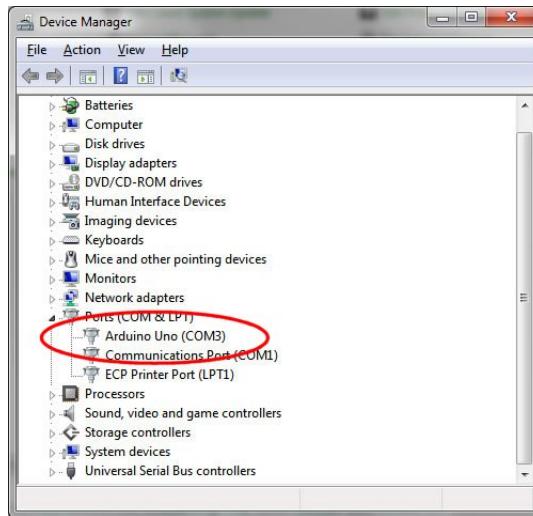


Figura 2.3: Arduino Uno en Administrador de Dispositivos de Windows.

Luego al abrir el IDE Arduino, podemos identificar los elementos mostrados en la figura 2.4 y en la barra de menús superior, se debe escoger el modelo de placa con que estemos trabajando y el puerto COM al que está conectada. (figura 2.5)

Una vez realizado esto, podemos comenzar a programar nuestra Arduino.

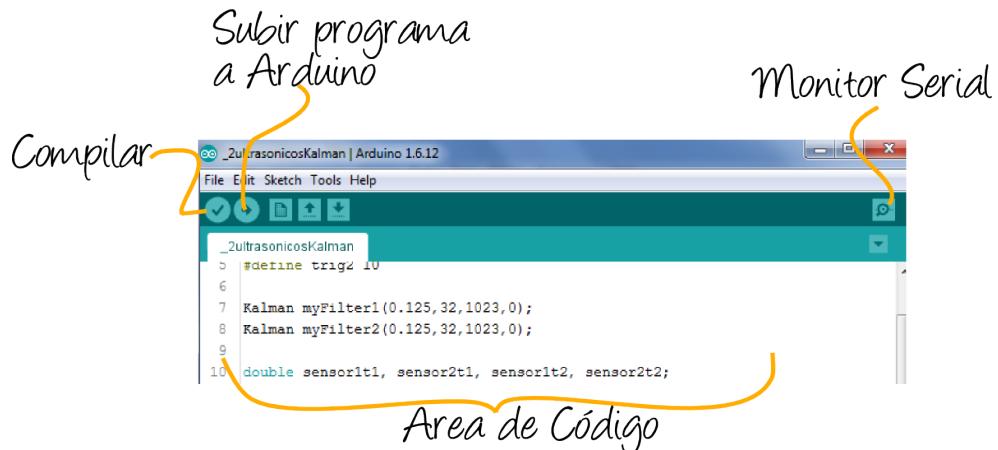


Figura 2.4: Elementos principales del IDE Arduino.

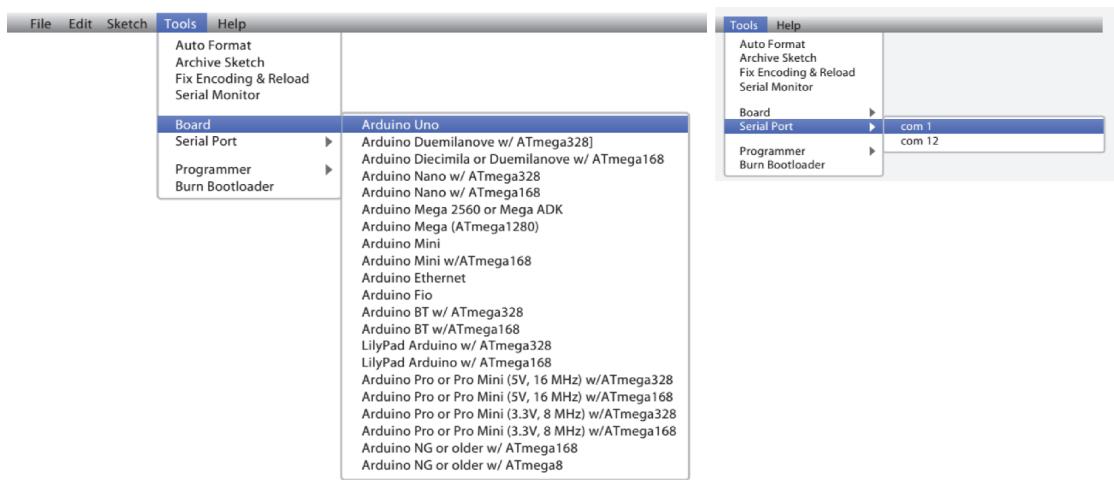


Figura 2.5: Configuración de Modelo de Placa y Puerto COM.

2.2.3. Estructura de un Código Arduino

Como ya se mencionó antes, el Lenguaje Arduino, no es más que una API desarrollada en C++, por lo tanto, los fundamentos de programación estructurada en dicho lenguaje, se mantienen, por ejemplo, declaración de librerías, directivas al compilador, declaración de variables e incluso la mayoría de los tipos de variables en C se mantienen, de hecho Arduino utiliza un compilador cruzado GCC para arquitecturas Avr de Atmel.

La principal diferencia radica en un nuevo tipo de estructura en el código, que se separa en dos funciones principales de tipo void que son las siguientes:

- **setup()**

Función que se ejecuta una sola vez al encendido del microcontrolador, aquí se deben configurar entradas, salidas, periféricos y todo aquello que se desee realizar una sola vez para todo el tiempo de funcionamiento del sistema.

- **loop()**

Bucle infinito que realizará repetidamente las instrucciones que se escriban dentro de él, originalmente está implementado con un `while(1)`. Aquí se realizan las operaciones principales del microcontrolador, como lectura de sensores, envío de datos y todo aquello que se quiera que esté realizando constantemente el microcontrolador.

2.3. Control de Salidas Digitales

A continuación, se muestra un código que sería el equivalente a un *Hola Mundo* en computación física. Este se puede encontrar dentro de la gran colección de ejemplos que trae el software. (`File>Examples>Basics>Blink`)

```
1 void setup() {
2   pinMode(13, OUTPUT);
3 }
4
5 void loop() {
6   digitalWrite(13, HIGH);      // turn the LED on (HIGH is the voltage level)
7   delay(1000);                // wait for a second
8   digitalWrite(13, LOW);       // turn the LED off by making the voltage LOW
9   delay(1000);                // wait for a second
10 }
```

Este código, pone a parpadear un Led integrado en la placa arduino, el cual está internamente conectado al Pin 13 de la tarjeta.

De aquí podemos analizar las instrucciones utilizadas:

- **pinMode(13,OUTPUT)**

Declara el pin número 13 del arduino como salida. Si cambiamos el número 13 por otro entre 0 y 12, la declaración afectará al pin correspondiente. De la misma manera, si se quiere que el pin sea entrada, basta cambiar `OUTPUT` por `INPUT`.

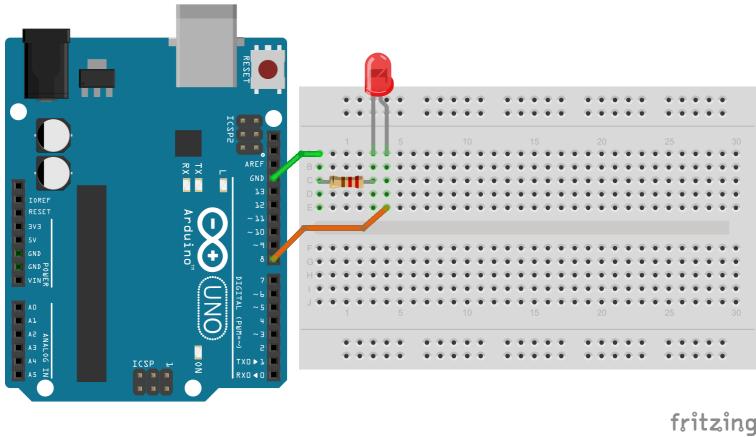


Figura 2.6: Circuito para encender un led conectado al pin 8.

- `digitalWrite(13,LOW)`

Como su nombre lo dice, realiza la escritura digital de un estado LOW en el pin 13. Cambiar el número afectará a otro pin, y para estados altos usamos la palabra HIGH.

- `delay(1000)`

Genera un retardo de 1000 **mili-segundos** después de la instrucción anterior. Si cambiamos ese número, esa será la cantidad de milisegundos que el microcontrolador quedará *paralizado*. Cabe notar que el micro no puede ejecutar ninguna otra tarea durante el delay. Para retardos del orden de micro-segundos se utiliza la instrucción `delayMicroseconds(retardo)`.

2.3.1. Actividad

1. Cargue el programa en inicial en el microcontrolador y verifique su funcionamiento físico.
2. Juegue con valores extremadamente pequeños para el delay y comente acerca del efecto visual que se produce.
3. Realice el montaje del circuito mostrado en la figura 2.6.
4. Modifique su código para hacer funcionar el circuito montado.

Cabe notar que la imagen del circuito, fue desarrollada en el software open-source *Fritzing*, en donde se muestra una protoboard con los componentes posicionados en ella. A nivel profesional, los circuitos no se muestran de esa manera sino que se representan por medio de diagramas como el mostrado en la figura 2.7 y puede ir de muchas formas sobre la protoboard que no es más que un conjunto de conductores ordenados. (véase la figura 2.8 tomada de [4])

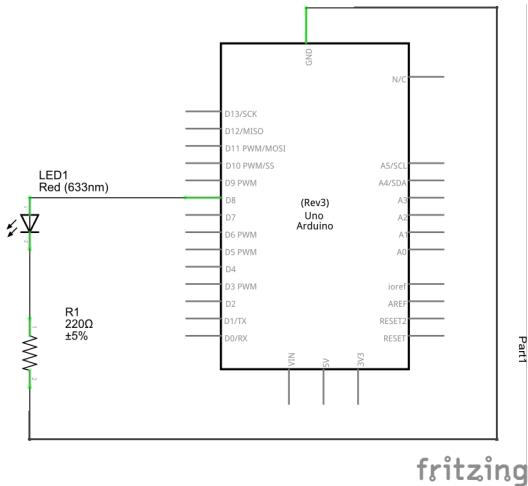


Figura 2.7: Diagrama de circuito de la Actividad 1.

2.4. Control de entradas Digitales

Entrada digital, quiere decir que nuestra Arduino será capaz de monitorear una señal eléctrica y realizar alguna acción en cuanto esto ocurra. La instrucción clave aquí es la siguiente:

- **digitalRead(3)**

Retorna el valor del estado del pin 3, que puede ser HIGH o LOW e incluso ser interpretado como 0 o 1. Se debe considerar declarar previamente el pin como entrada.

A continuación, se presenta un código, que cambia el estado del led del circuito de la imagen 2.9 al ser presionado un botón, es decir, si el led está apagado, al presionar el botón encenderá y al presionar nuevamente, se apagará.

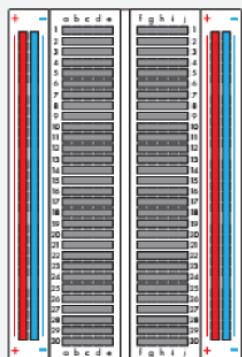
En este código, se puede ver como es posible utilizar características y técnicas del lenguaje C, también resulta de interés el uso de la instrucción `delay(50)`, puesto que el microcontrolador trabaja a una velocidad tal que, es posible que reaccione a los rebotes que produce el botón al ser presionado, aunque nosotros no lo notemos.

Breadboard

1 Vertical Connection (+ Power and - Ground // See Diagram Below)

2 Horizontal Connection (a-e & f-j // See Diagram Below)

How's it all connected?



+ Power:

Each + sign runs power anywhere in the vertical column.

- Ground:

Each - sign runs to ground anywhere in the vertical column.

Horizontal Rows:

Each of these rows numbered 1-30 are comprised of five horizontal sockets. Components placed in the same row will be connected in a circuit when power is running.

Making a Connection:

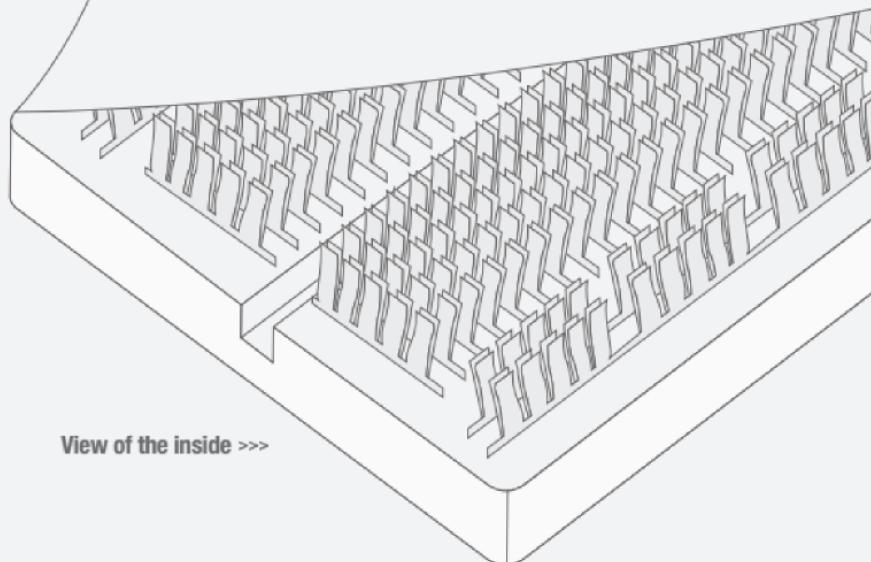
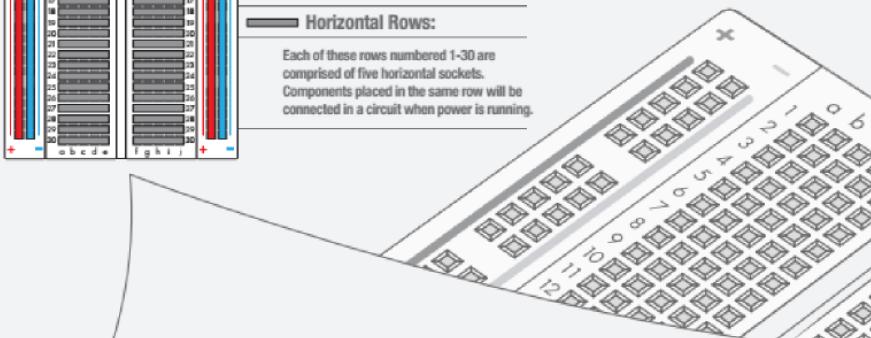
Above the breadboard



CONNECTED!



Inside the breadboard



View of the inside >>

Figura 2.8: Infografía acerca de la protoboard. [4]

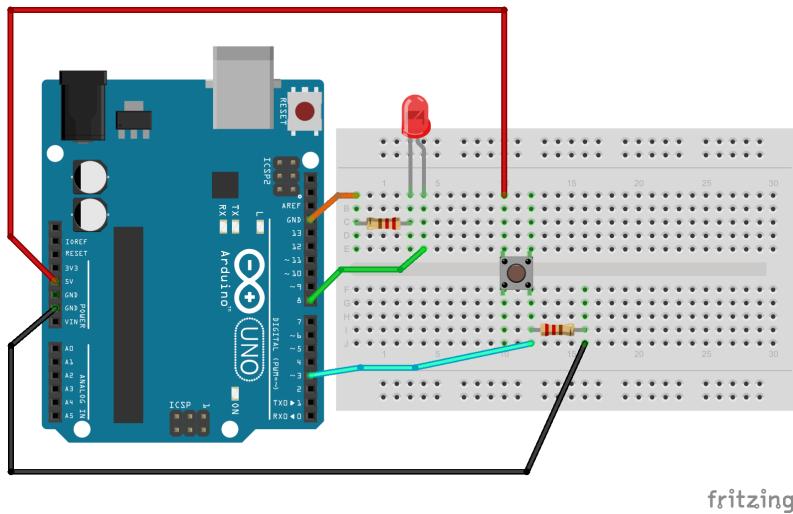


Figura 2.9: Circuito para probar entrada digital del pin 3.

```

1 boolean estadoLed = false;
2
3 void setup() {
4
5   pinMode(8, OUTPUT); //declaro el pin 8 como salida
6   pinMode(3, INPUT); //declaro el pin 3 como entrada
7
8 }
9
10 void loop() {
11   //pregunto si se ha presionado el boton en el pin 3
12   if(digitalRead(3)){
13
14     delay(50); //espero a que el boton deje de rebotar
15     estadoLed=!estadoLed; //cambio el estado del flag
16
17   }
18
19   digitalWrite(8,estadoLed); //escribo el valor en el pin correspondiente
20   //al led
21
22 }
```

2.4.1. Actividad

1. Con ayuda de su compañero/a, realice el montaje del circuito mostrado en la figura 2.9.
2. Cargue el programa correspondiente en su Arduino.
3. Cree un nuevo programa, tal que al presionar el botón, el led parpadee tres veces.

2.5. Control de Salidas PWM

Pulse Wide Modulation

Tal como se vio en un principio, que comportamientos extraños ocurrían al exponer el led del pin 13 a encendidos y apagados demasiado rápidos, sabemos de cierta forma que controlar los tiempos de encendido y apagado de una señal eléctrica a gran velocidad, es a grandes rasgos, controlar su potencia.

Beneficiosamente, Arduino permite ejercer un control directo sobre la potencia *virtual* de los pines marcados con el símbolo \sim , mediante la técnica conocida como *PWM Modulación por Ancho de Pulso* que consiste en controlar el balance entre los tiempos de estado alto y bajo de la señal.

La instrucción que permite realizar estas operaciones es la siguiente:

- **analogWrite(pin,potencia)**

Permite que el pin correspondiente, emita una señal a la potencia especificada como un número entre 0 y 255. Por ejemplo.

- `analogWrite(9,255); //Pondrá el equivalente a 5 volts en el pin 9.`

- `analogWrite(5,1); //Pondrá el equivalente a un voltaje muy pequeño en el pin 5.`

En la figura 2.10, se muestran las formas de onda aplicada a los pines de salida, normalmente esto ocurre a una frecuencia de 32kHz.

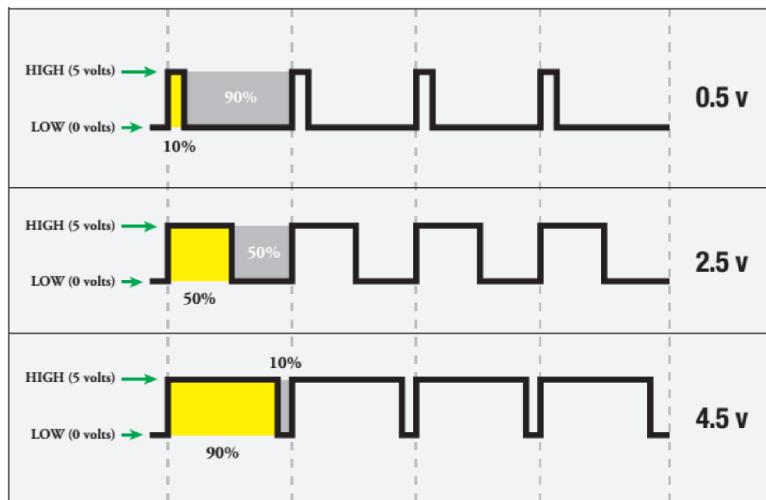


Figura 2.10: Señales PWM.

Aunque la instrucción hace alusión a una señal analógica, resulta evidente que es solamente una aproximación digital.

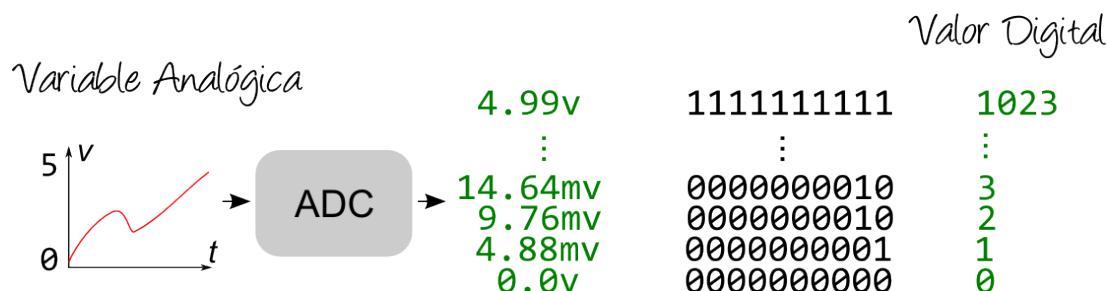
2.5.1. Actividad

1. Utilizando el mismo circuito de la figura 2.9, programe el siguiente código, que permite probar la instrucción `analogWrite`, dejando el led encendido a distintos valores de potencia.

```
1 void setup() {  
2  
3     pinMode(9, OUTPUT); //declaro el pin 9 como salida  
4  
5 }  
6  
7 void loop() {  
8  
9     analogWrite(9,100); //enciendo el pin 9 con un valor de potencia  
    determinada  
10}  
11}
```

2. Utilice sus habilidades en programación, para crear un algoritmo que varíe la intensidad del led constantemente de manera continua, como si *respirara*. Recuerde que las sentencias condicionales, son las mismas del lenguaje C.

2.6. Lectura de Entradas Analógicas



Arduino, está dotado de un conversor analógico-digital con una profundidad de 10 bits, es decir, que es capaz de transformar magnitudes analógicas entre 0 y 5 volts, en números de 0 a 1024 *cuentas*. Lo anterior permite establecer que la resolución de voltaje de Arduino, es de 4.88 mili volts, lo que significa que es ese el mínimo cambio de voltaje que el sistema puede detectar.

La instrucción que nos permite capturar el número equivalente al voltaje medido es la siguiente:

- `analogRead(A3)`

Retorna el número de cuentas equivalente al voltaje presente en el pin A3. En total Arduino cuenta con 6 entradas analógicas, tal como se vio en la figura 1.2. Estos pines, no necesitan ser declarados como entradas.

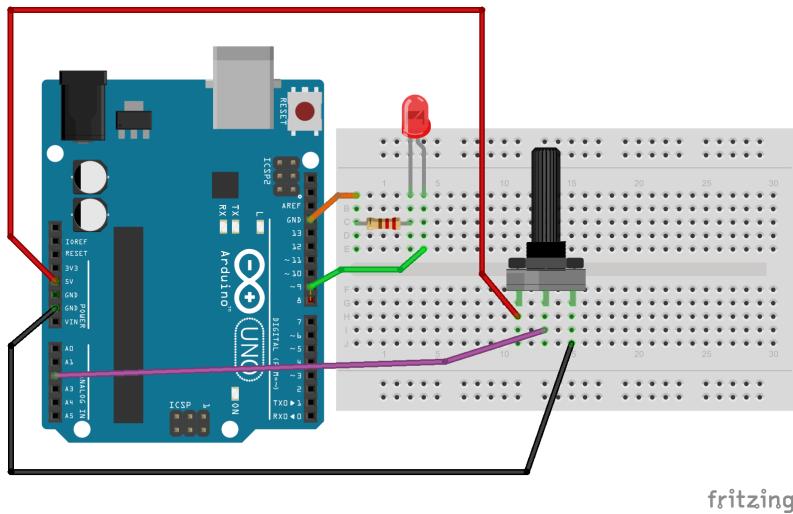


Figura 2.11: Circuito de prueba de la actividad.

2.6.1. Actividad

A continuación, se presenta un programa y circuito, que lee el voltaje analógico entregado por un potenciómetro en el pin A2 y lo almacena en la variable tipo entera `medicion`. Se pide que usted realice lo siguiente:

1. Realice el montaje del circuito mostrado en la imagen 2.11.
2. Cargue el siguiente programa en su Arduino.

```

1 int medicion;
2
3 void setup(){
4
5 }
6
7 void loop(){
8     medicion=analogRead(A2);
9 }
```

3. Modifique el programa presentado, de tal manera que **si el valor medido, sobrepasa las 500 cuentas, se debe encender el led**.

Cabe notar que en este caso, el encendido del led será una alarma indicadora del sobrepaso del la medición por algún valor determinado.

2.7. Comunicación Serial

En Arduino, la comunicación serial, tiene la función principal de cargar programas al microcontrolador, pero tambien puede ser usada para enviar y recibir mensajes en forma de texto ASCII entre el PC, la tarjeta e incluso

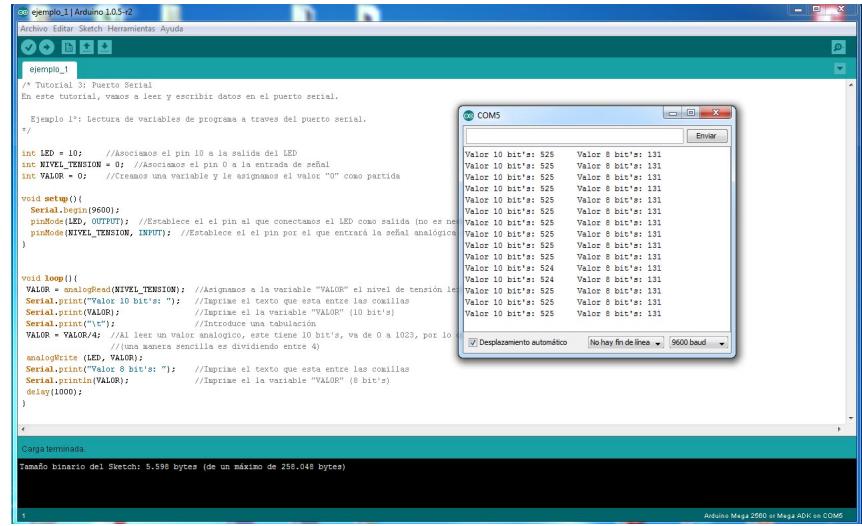


Figura 2.12: Ventana del Monitor Serial.

algún otro dispositivo que permita este tipo de comunicación, cómo modulos de comunicación y GPS's, entre otros.

Una de las maneras más simples de experimentar con la comunicación serial, es el envío y recepción de mensajes a través del *Monitor Serial* que el software trae incorporado.

Las instrucciones relevantes, para manejar este tipo de comunicación son:

- **Serial.begin(9600)**

Inicia la comunicación serial a la velocidad en baudios³ especificada. La práctica habitual es utilizar esta instrucción en el *setup()*.

- **Serial.print("mensaje")**

Envía la palabra *mensaje* desde Arduino al PC, mostrándose en el Monitor Serial.

- **Serial.println("mensaje")**

Realiza lo mismo que la instrucción anterior, pero esta vez añade un retorno de carro al final del mensaje, es decir que si se escribe luego de esta instrucción, los mensajes continuarán en la línea siguiente, no así en el caso anterior, donde los mensajes se van acumulando hacia el lado.

Cabe notar que ambas instrucciones, aceptan como argumentos, variables tipo *int*, *float*, entre otras y muestran su valor resultante en como caracteres ASCII.

- **Serial.available()**

Retorna la cantidad de bytes almacenados en el buffer de entrada. Básicamente si este valor es distinto de cero, quiere decir que están llegando datos por el puerto serial en Arduino.

- **Serial.read()**

Retorna el primer byte disponible en el buffer, una vez leído pasa al siguiente byte. Cabe notar que los datos vienen en binario correspondiente al código ASCII del carácter entrante.

³Bits por Segundo

- `Serial.parseInt()`

Retorna el valor entero que se pueda identificar en el buffer de entrada, resulta muy útil para leer variables de entrada y almacenarlas como dato entero. Cabe notar que cuando no recibe datos retorna el valor entero cero.

Existen más instrucciones para el puerto Serie y se pueden encontrar en el sitio de referencia para la clase Serial del en la página oficial de Arduino. [5]

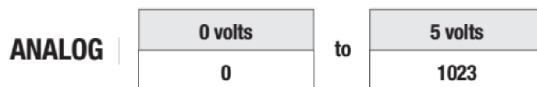
2.7.1. Actividad: Lectura de variable analógica desde el Puerto Serie

A continuación, presenta un código que permite leer valor del potenciómetro de la actividad anterior, almacenarlo en un variable de tipo entera y enviar ese valor de manera que se pueda visualizar a través del Monitor Serial.

```
1 int medicion;
2
3 void setup(){
4   Serial.begin(9600);
5 }
6
7 void loop(){
8   medicion=analogRead(A2);
9
10  Serial.print("Valor Medido: ");
11  Serial.println(medicion);
12  delay(200);
13 }
```

Cabe notar, que en el código se utiliza un `delay` para no saturar el Monitor Serial con mensajes a gran velocidad.

1. Cargue el programa mostrado en su Arduino y mueva el potenciómetro para verificar el correcto funcionamiento de la aplicación.
2. Modifique el programa anterior, de manera que se muestre a través del Monitor Serial el valor de **voltaje** que se está midiendo en la entrada.



En instrumentación industrial, esta operación se conoce como *Escalamiento* y es habitual que los sistemas de control como PLC's la realicen, puesto que la mayoría de los instrumentos de medición entregan una señal en corriente de 4 a 20ma. la que debe ser transformada a la variable original de medición.

2.7.2. Actividad: Envío de instrucciones a Arduino a través del Puerto Serie

A continuación, presenta un código que permite enviar una instrucción a Arduino a través del Puerto Serie, de manera que se le ordene el encendido o apagado de un Led. Se utiliza el mismo circuito de la figura 2.11.

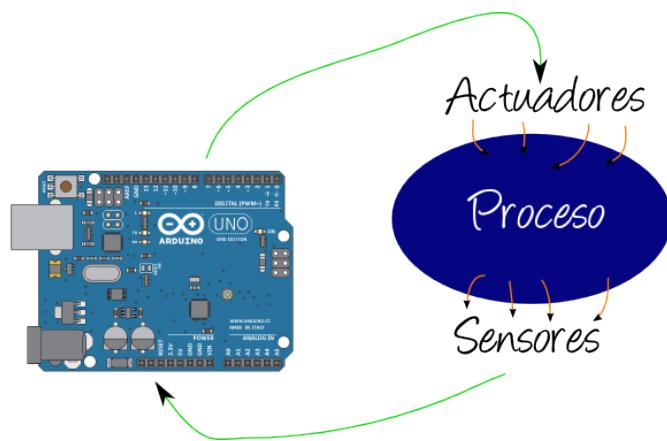
```
1 int byteEntrante;           // aqui guardamos el valor que
2                     //entra en el puerto serie
3
4 void setup()
5 {
6     //se inicia la comunicacion Serial
7     Serial.begin(9600);
8     //se declara el pin para el led como salida
9     pinMode(9, OUTPUT);
10}
11
12 void loop()
13 {
14     // se esperan datos en el puerto serie
15     if (Serial.available() > 0)
16     {
17         // leemos el primer byte del buffer
18         byteEntrante = Serial.read();
19         // si llega una H encendemos el led
20         if (byteEntrante == 'H')
21         {
22             digitalWrite(9, HIGH);
23         }
24         // si llega una L lo apagamos
25         if (byteEntrante == 'L')
26         {
27             digitalWrite(9, LOW);
28         }
29     }
30 }
```

1. Cargue el programa mostrado en su Arduino y verifique el correcto funcionamiento de la aplicación.
2. Modifique el programa anterior, de manera que nos confirme a través del mismo puerto serial si el led está encendido o apagado y si enviamos algún carácter no esperado, nos indique la situación y nos recomiende ingresar H o L.

Capítulo 3

Lectura de Sensores y Control de Actuadores con Arduino.

Una de las características de los sistemas como arduino, es que pueden interactuar de manera física con el mundo que nos rodea, sus creadores lo llaman *Physical Computing*. Toma mediciones del entorno o proceso por medio de sensores, realiza algún tipo de procesamiento o toma de decisión y en base a ello ejerce alguna acción sobre el sistema por medio de los actuadores.



En este capítulo, veremos algunos casos particulares de sensores y actuadores que se pueden utilizar con esta tarjeta ya sea por medio de mediciones analógicas o digitales.

3.1. Sensor de luminosidad LDR

LDR significa *Light Dependant Resistor* por sus siglas en inglés, y se trata de un sensor de luz; este cambia su resistencia eléctrica dependiendo de la cantidad de luz que incida sobre él. También es común que se los llame *Foto-resistencia* debido a este mismo principio.

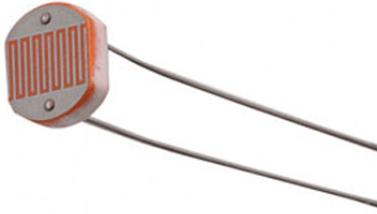


Figura 3.1: Sensor de Luminosidad LDR.

En la figura 3.1, se puede ver como luce este tipo de sensor. A mayor luz, el LDR disminuirá su valor resistivo, mientras que en la oscuridad presentará una gran resistencia eléctrica.

Como ya se ha visto antes, arduino sólo mide voltajes, ya sea de forma digital o analógica y no mide resistencia. Gracias a ley de Ohm, estas magnitudes están ligadas y es posible generar voltajes proporcionales a las mediciones de luz por medio de un *circuito divisor de tensión*, como el que se muestra en la figura 3.2. Cabe notar, que este principio de medición, es válido para cualquier sensor que sea del tipo resistivo, por ejemplo, sensores flex o galgas extensiométricas, sensores de carga, etc.

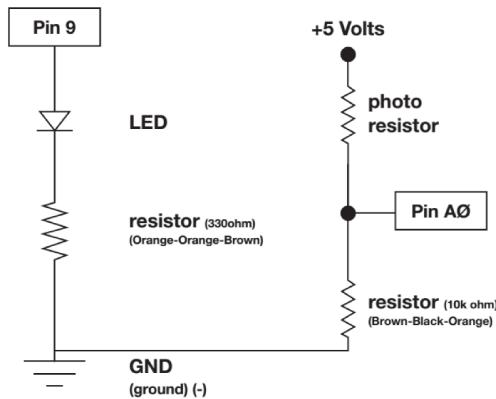


Figura 3.2: Divisor de tensión para medir luminosidad con un LDR.

En este caso, el Led conectado al pin 9, no es necesario para la medición con LDR, pero luego programaremos para que nos indique si el nivel de luz ha bajado de algún valor determinado.

3.1.1. Actividad

1. Realice el montaje en protoboard, del circuito de la figura 3.2. Esta vez no se incluye el diagrama de la protoboard, pues debe ser usted quien decida el montaje.
2. Modifique el código de la Actividad 2.7.1 para poder verificar los distintos valores entregados por el sensor ante diversas condiciones de Luz, puede tapar el sensor, iluminarlo con la linterna de su teléfono y revisar su comportamiento.
3. Defina niveles de Luminosidad para una sala iluminada y una sala oscura, en base a eso, programe el Arduino para que encienda automáticamente un led al detectar que la sala está oscura.

3.2. Sensor de Temperatura LM35

El sensor LM35, está diseñado para medir temperaturas entre 0 y 150°C aproximadamente, internamente contiene circuitería analógica perfectamente calibrada para asegurar una precisión máxima de 0.5°C dentro de ese rango.

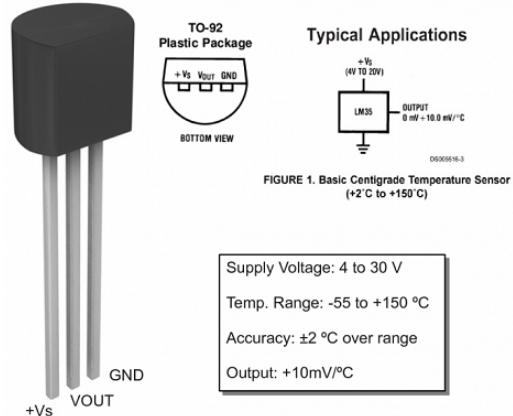


Figura 3.3: Sensor LM35, pines del dispositivo.

A su salida, entrega una señal de voltaje proporcional a la medición de temperatura con característica de **10mV por cada Grado Celsius** según su hoja de especificaciones técnicas. [6]

En este caso, es necesario realizar el escalamiento, para que en el puerto serial nos muestre el valor de temperatura en vez del número de cuentas correspondiente a la medición.

3.2.1. Actividad

1. Considerando que el sensor entrega una medida equivalente 10 mili-volts por cada grado medido, esto es 0 volts a 0°C y a los 5 volts estaría mostrando 500°, matemáticamente hablando; realice el cálculo que debe ser implementado en arduino para que este muestre a través del puerto serial, el valor de temperatura medido.
2. Conecte el sensor de temperatura a la entrada analógica A0 de Arduino de acuerdo a las especificaciones del fabricante—figura 3.3—, registre también el diagrama de dicha conexión.
3. Programe el microcontrolador, de manera que al abrir el puerto serial este muestre los valores de temperatura como sigue:

TEMPERATURA: 23.0 Grados
TEMPERATURA: 23.5 Grados

4. Establezca un indicador de alarma al sobrepasar los 50°C, mediante un Led conectado al Pin 9 de su Arduino.

3.2.2. Serial Plotter

El Software Arduino Incorpora en sus últimas versiones una utilidad llamada *Serial Plotter*, que permite graficar variables en tiempo Real, en la pantalla del ordenador. Serial Plotter, necesita recibir números separados por comas, para poder mostrarlos, si recibe texto, no mostrará nada. Pruebe el Serial Plotter con el sensor LM35 y el último programa de la actividad.

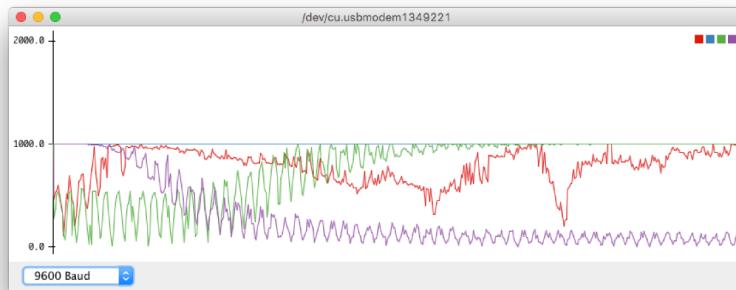


Figura 3.4: Serial Plotter de Arduino IDE.

3.3. Sensor Ultrasónico Parallax PING

El sensor ultrasónico Ping, permite medir distancias mediante el envío de un pulso de ultrasonido y el conteo del tiempo de vuelo del pulso hasta recibarlo de vuelta.



Figura 3.5: Sensor Ultrasónico Parallax Ping.

Al tener un objeto sólido en frente, la onda ultrasónica rebotará y será recibida con una diferencia de tiempo proporcional a la distancia a la que se encuentra el objeto, cabe notar que el pin que se usa tanto como para emitir, como recibir el ultrasonido es el mismo, por lo tanto el microcontrolador debe ser capaz de cambiar su configuración de entrada a salida durante la marcha. Este sensor fue desarrollado inicialmente para complementar un kit de robótica educacional llamado *Boe-Bot* basado en el microcontrolador *Basic Stamp*, hoy en día es usado muchas aplicaciones que van desde la robótica hasta la IoT ya que es perfectamente compatible con Arduino.

La realizar mediciones reiteradas con este sensor, requerirá que el controlador execute por lo menos los

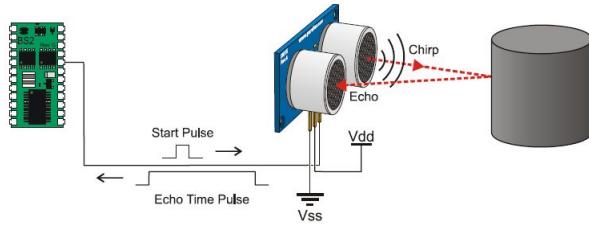


Figura 3.6: Medición con Sensor Ping y Microcontrolador Basic Stamp

siguientes pasos, de forma repetitiva:

1. Declarar el pin **Sig** como **Salida**.
2. Emitir un pulso con una duración de 5 micro segundos.
3. Declarar el pin **Sig** como **Entrada**.
4. Contar el tiempo hasta que se recibe el pulso ultrasonico y guardarlo en una variable, este sera el *Tiempo de Vuelo*.

Finalmente, el tiempo de duración será proporcional a la distancia. En base a la velocidad del sonido de 340 metros por segundo o 29 microsegundos por centímetro, es posible establecer que el pulso ultrasónico va y vuelve, luego para encontrar la distancia hasta el objeto, necesitamos la mitad de la distancia recorrida.

```
distancia = tiempoVuelo / 29 / 2;
```

A continuación, se presenta un código que toma la medición del sensor Ping y muestra el valor de la distancia en centímetros a través del puerto serial.

```

1 long tiempoVuelo, distancia;
2 const int pingPin = 7; // pin Sig del sensor
3
4
5 void setup() {
6     // inicio de la comunicacion Serial
7     Serial.begin(9600);
8 }
9
10 void loop() {
11
12     // Se declara el pin como salida
13     pinMode(pingPin, OUTPUT);
14
15     // Se forma el pulso de 5 microsegundos
16     digitalWrite(pingPin, LOW);
17     delayMicroseconds(2);
18     digitalWrite(pingPin, HIGH);
19     delayMicroseconds(5);
20     digitalWrite(pingPin, LOW);

```

```

21 // se declara el pin como entrada
22 pinMode(pingPin, INPUT);
23
24 tiempoVuelo = pulseIn(pingPin, HIGH);
25
26 // calculamos el valor en CM
27 distancia = tiempoVuelo / 29 / 2;
28
29 // se muestra el resultado
30 Serial.print("Distancia:"); 
31 Serial.println(distancia);
32
33 delay(100);
34
35 }

```

3.3.1. Actividad

1. Realice el conexionado del sensor a su tarjeta arduino.
2. Programe el microcontrolador y verifique el correcto funcionamiento de la aplicación.
3. Modifique el código de forma que pueda ser utilizado con el Serial Plotter. Comente acerca de la fidelidad de la señal mostrada y el ruido presente en esta.
4. Implemente un código, tal que cada 10 mediciones arduino muestre el promedio de estas, con el fin de obtener una medición más suave.

3.4. Pantalla LCD 16x2

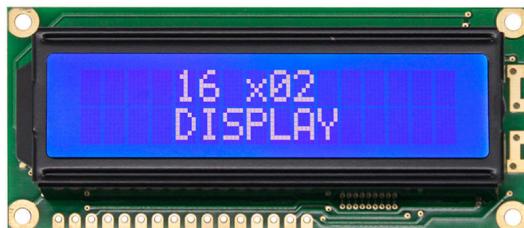


Figura 3.7: Pantalla Lcd 16x2.

Se trata de un periférico bastante popular, que aún se puede encontrar en algunos equipos comerciales e industriales. Por ejemplo, estas máquinas expendedoras, de Inacap Concepción.

Estas pantallas utilizan un driver HD44780 creado el año 1987, en ese entonces, algo equivalente a lo que hoy sería el procesador de un monitor de ordenador.

El trabajo que hace este micro, no es menor, puesto que se encarga de encender o apagar cada uno de los pequeños cuadrados que componen cada carácter de nuestra pantalla. Una ardua tarea, si pensamos en que cada



Figura 3.8: Utilización de pantalla Lcd en Vending-Machine.

uno de estos puntos utilizaría una salida de nuestro un microcontrolador como Arduino, obviamente no sería posible.

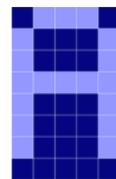


Figura 3.9: Matriz de puntos que forma un carácter.

El producto, consta de una interfaz sencilla, mostrada en la figura 3.10, es universal casi a todas las pantallas LCD 16x2 e incluso otras variantes como 16x4 o alguna otra basada en el mismo driver.

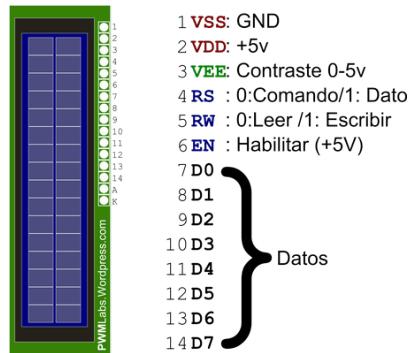


Figura 3.10: Interfaz de conexión de una pantalla Lcd 16x2.

Aunque, son muchos pines, la conexión es bastante simple, y podemos dividirla, en:

- **Pines Alimentación:** VSS, VDD.
- **Contraste:** Conectamos un Potenciómetro de 10k y con el manipulamos para una pantalla más clara u oscura.

- **Control:** RS y EN son los pines que usará Arduino para manejar la pantalla. RW sirve para enviar el dato de vuelta, pero en este caso eso es innecesario así que se conecta a GND.
- **Datos:** En este Byte, Arduino, enviará los datos o comandos. Aunque son 8 líneas desde el nuestra tarjeta hasta la pantalla, nuestra librería funciona sólo con 4 bits. D4 , D5, D6 y D7, por lo tanto, los demás pines se aterrizan.

En resumen, además de la alimentación, el contraste y aterrizar los pines que no se utilizan; se deben conectar a salidas de Arduino, los pines RS, EN, D4, D5, D6 y D7.

Enviar los datos ASCII en hexadecimal hasta la pantalla resulta una tarea un tanto ardua de programar, por lo cual, se hace uso de una librería de terceros, en este caso se utilizaremos LiquidCrystal, que en las últimas versiones ha pasado a ser parte de las librerías oficiales que Arduino trae en sus repositorios.

3.4.1. Instalación de Librerías

Para instalar una nueva librería en Arduino IDE, debe dirigirse a Sketch>Include Library>Manage Libraries, como se muestra en la figura 3.11.

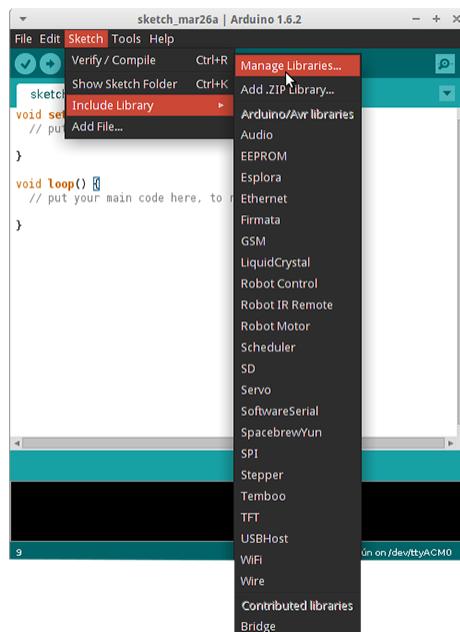


Figura 3.11: Acceso al *Library Manager* en Arduino IDE.

Una vez abierta la ventana *Library Manager*. Se despliega un listado con las librerías añadidas oficialmente a los repositorios de Arduino, en el buscador agregamos *LiquidCrystal* y debería aparecer nuestra librería con un botón para instalar, presionamos dicho botón y seguimos el proceso. En la figura 3.12, se muestra la ventana del *Library Manager*, que en este caso no muestra el botón para instalar *LiquidCrystal*, pues ya está instalado en ese equipo.

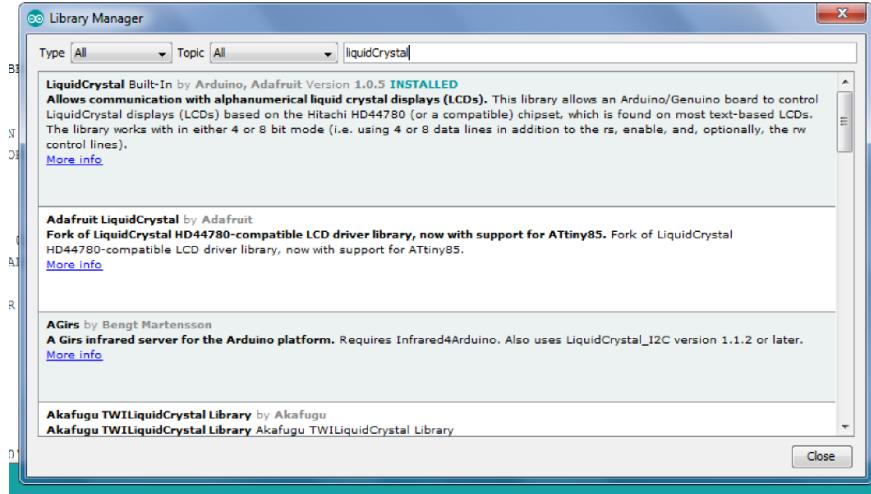


Figura 3.12: Ventana del *Library Manager*.

3.4.2. Programación de Pantalla LCD

A continuación, se provee un programa de ejemplo que muestra en la pantalla Lcd el mensaje "Hello World" y bajo este un contador que indica los segundos transcurridos desde el inicio del programa.

La instrucción `millis()`, retorna la cantidad de milisegundos que han transcurrido desde el último reset o encendido de la tarjeta, esta instrucción resulta muy útil para operaciones que requieren contar o controlar el tiempo en que se ejecuta una actividad.

```

1 // declaracion de pines para el LCD
2 const int rs = 12, en = 11, d4 = 5, d5 = 4, d6 = 3, d7 = 2;
3
4 // Creacion del objeto LCD
5 LiquidCrystal lcd(rs, en, d4, d5, d6, d7);
6
7 void setup() {
8     // Inicializacion de un LCD 16 columnas y 2 filas
9     lcd.begin(16, 2);
10
11    // Escribe el mensaje en la pantalla del LCD
12    // En la Posicion 0,0
13    lcd.print("hello ,world!");
14 }
15
16 void loop() {
17     // ponemos el cursor en la columna 0, fila 1
18     // linea 1 es la segunda fila, comienzan en 0:
19     lcd.setCursor(0, 1);
20
21     // muestra un conteo de tiempo:
22     lcd.print(millis() / 1000);

```

3.4.3. Actividad

- Realice la conexión de la pantalla Lcd a la placa Arduino de acuerdo a la figura 3.13, no olvide considerar el potenciómetro de $10k\Omega$, este le permitirá ajustar el brillo de la pantalla.

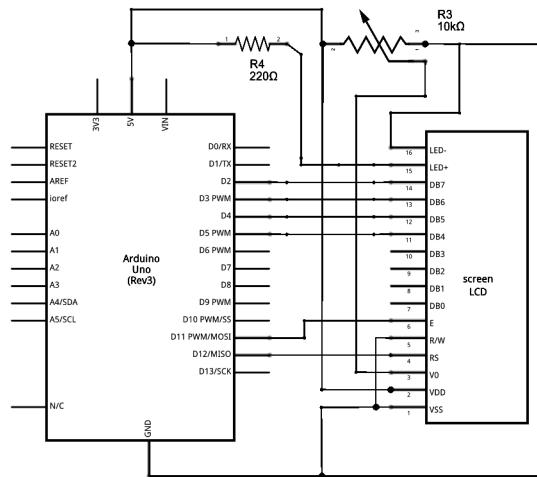


Figura 3.13: Conexión de Pantalla Lcd a Arduino.

- Cargue el programa de la subsección 3.4.2 en su Arduino y verifique el correcto funcionamiento del ejemplo.
- Modifique el código y circuito de forma que esta vez se pueda mostrar en pantalla, la medición de algún sensor de su elección, de los vistos en este curso. (véase el ejemplo de la figura 3.14)

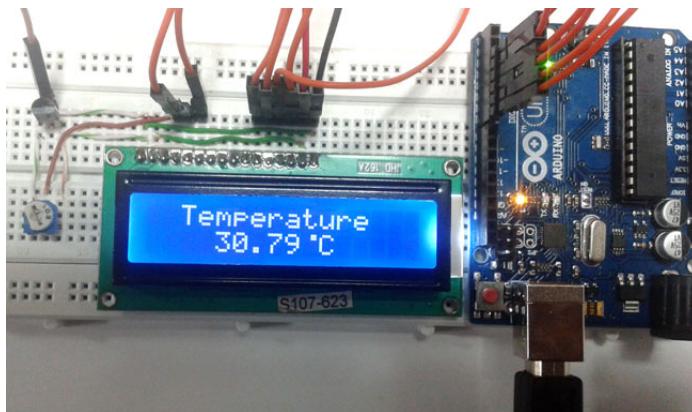


Figura 3.14: Ejemplo de aplicación con Sensor LM35 y Pantalla Lcd.



Figura 3.15: Interior de un Servomotor.

3.5. Control de Servomotores

El concepto de **Servo-Motor**, consiste en un sistema de control rotacional de velocidad o posición, que se caracteriza habitualmente, por traer integrado en un sólo producto –figura 3.15–, al menos los siguientes elementos:

1. Motor DC.
2. Caja reductora.
3. Sensor de Posición o Velocidad.
4. Circuito analógico de control proporcional.

Al contrario de cualquier motor convencional, que gira libremente en algún sentido al ser alimentado, estos motores pueden entregar con mediana precisión, ángulos o velocidades determinadas, en base a una señal de comando, que generalmente consiste en un pulso de una duración determinada. Los *servos* que entregan posición, se conocen como *Standar Servo*, los que entregan velocidad, como *Continuous Servo* y son la base de la Robótica y el Control de Movimiento tanto nivel, científico e industrial como educacional.

La conexión de estos servomotores, responde a un estándar respetado por la mayoría de los fabricantes, los cables Rojo y Negro corresponden a la alimentación, generalmente hasta 12 volts y el cable Blanco corresponde a la señal de comando, que en este caso, obtendremos de un pin de nuestra Arduino.

Para la programación, se utilizará la librería *Servo*, contenida en el listado oficial y que puede ser instalada según el procedimiento de la subsección 3.4.1 de la sección 3.4.

A continuación, se muestra un ejemplo de código que permite que nuestro motor se mueva secuencialmente entre 0, 90 y 180 grados.

```

1 #include <Servo.h>
2
3 // Declaramos el objeto servo
4 Servo servoMotor;
5
6 void setup() {
7     // se inicializa el servo en el pin 9
8     servoMotor.attach(9);
9 }
10

```

```

11 void loop() {
12
13     // posicion 0 grad
14     servoMotor.write(0);
15     // Esperamos 1 segundo
16     delay(1000);
17
18     // posicion 90 grad
19     servoMotor.write(90);
20     // Esperamos 1 segundo
21     delay(1000);
22
23     // posicion 180 grad
24     servoMotor.write(180);
25     // Esperamos 1 segundo
26     delay(1000);
27 }
```

3.5.1. Actividad

1. Realice el montaje del circuito mostrado en la figura 3.16 que permite accionar un servomotor.

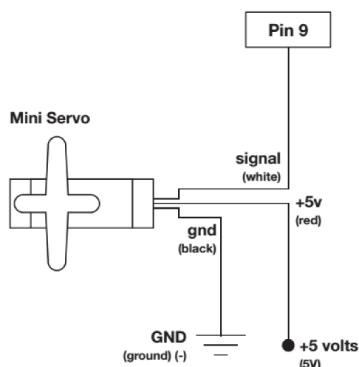


Figura 3.16: Conexión de un ServoMotor.

2. Cargue el programa mostrado anteriormente y verifique su funcionamiento.
3. Realice las modificaciones necesarias, para hacer que el servomotor se mueva a partir del ángulo de un potenciómetro. Una instrucción útil para esto:

- `map(input, minIn, maxIn, minOut, maxOut);`
Retorna un entero que permite escalar la variable `input` que oscila entre los valores `[minIn, maxIn]`, entregando un resultado que oscila entre `[minOut, maxOut]`.

3.6. Motores DC



Figura 3.17: Motor DC de escobillas.

Los motores DC –de corriente continua–, que podemos encontrar en juguetes y ventiladores de cpu, como el de la figura 3.17, se caracterizan por girar inmediatamente después de conectarles una fuente de alimentación. Generalmente pueden funcionar a partir de los 3 voltios, pero por lo menos requieren sobre 100mA de corriente para funcionar, corriente que lamentablemente no puede ser entregada por ninguna de las salidas de nuestra Arduino.¹ Aquí enfrentamos un problema típico de la electrónica, que tiene que ver a grandes rasgos, con un área llamada **Control de Potencia**, que se encarga de acoplar señales de control como las que trabaja Arduino, con **corrientes de fuerza** como la que necesita un motor para poder mover alguna carga o su propio eje.

Nace entonces, la necesidad de contar con la circuitería necesaria para realizar el *acople* mencionado, generalmente este tipo de circuitos se conoce como *Driver*.

A continuación, en la figura 3.18, se muestra un driver básico hecho en base a un transistor 2n2222 que permite a Arduino manejar las corrientes que requiere el motor.

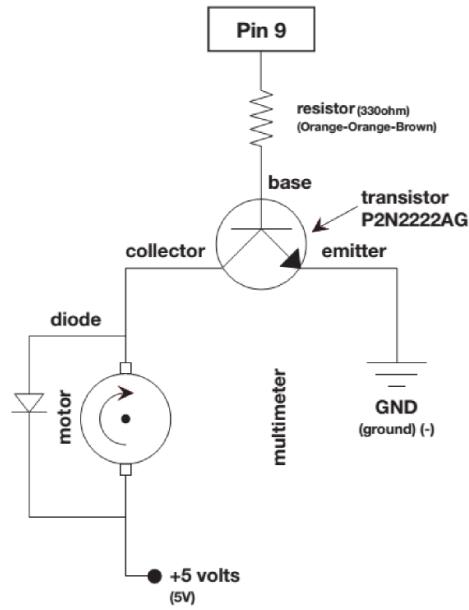


Figura 3.18: Driver para motor DC basado en Transistor 2n2222.

¹Atmel, recomienda un máximo de 20ma por salida para aplicaciones de prueba, en general se debe diseñar para que salga la mínima corriente posible del microcontrolador.

Cabe notar que en este circuito, se utilizan dos elementos desconocidos hasta ahora, los cuales se describen brevemente a continuación:

- **Transistor:** Elemento semiconductor, que a partir de una pequeña corriente en su terminal de base, puede manejar proporcionalmente corrientes entre sus terminales Colector y Emisor. El modelo 2n2222 se caracteriza por poder manejar una corriente de hasta 200ma, lo que resulta suficiente para nuestra aplicación.
- **Diodo:** Elemento que permite el paso de la corriente en un solo sentido, en esta aplicación permite bloquear corrientes producidas por el *chisporroteo* del motor que, de no estar el diodo, terminarían dañando el transistor.

3.6.1. Actividad

1. Realice el montaje del circuito mostrado en la figura 3.18. Con el fin de simplificar el montaje, se muestra de forma explícita la conexión en la figura 3.19.

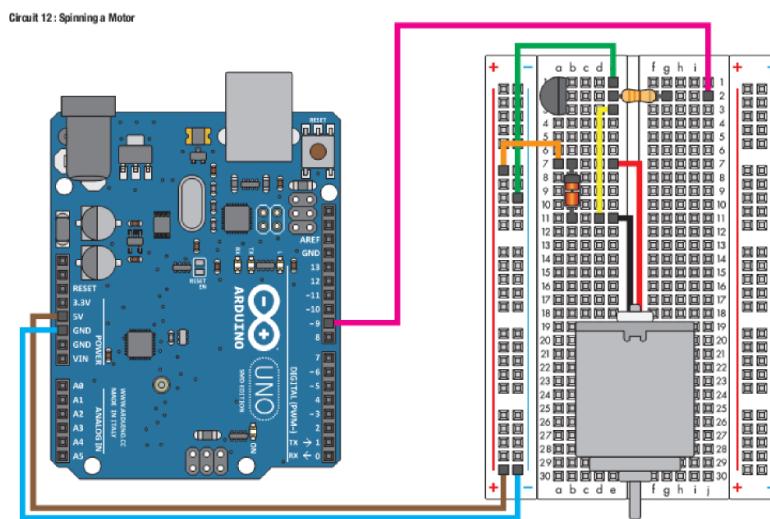


Figura 3.19: Driver para motor DC, tomado de [4].

2. Escriba un programa –o reutilice alguno anterior–, que haga girar el motor durante un segundo, luego que este se detenga durante un segundo y que esto se repita eternamente.
3. Escriba un programa que envíe distintas señales PWM al pin 9, comente acerca del funcionamiento que presenta el motor. (Revise la sección 2.5)

3.7. Cargas Fuertes y Corriente Alterna por medio de Relé

En ocasiones, puede ser necesario utilizar Arduino para comandar artefactos cotidianos, como encender una ampolleta de nuestra casa, lo cual no sería posible, puesto que nuestra tarjeta trabaja con 5 volts en corriente continua y en nuestra vivienda el estándar es 220 volts en corriente alterna, por lo que cualquier interconexión directa entre ambos terminaría *riendo* nuestra Arduino.

Bien se podría utilizar un transistor, pero no serviría el pequeño 2n2222 que utilizamos en prácticas anteriores, si no que en este caso, se requeriría calcular un driver que utilice arreglos de varios transistores con especificaciones para mayor potencia, los que en la actualidad son bastante caros y se utilizan mayormente en aplicaciones industriales como parte de costosos equipos, por lo cual descartamos este tipo de dispositivos para aplicaciones simples. Utilizaremos en cambio, un dispositivo bastante simple, llamado Relé.

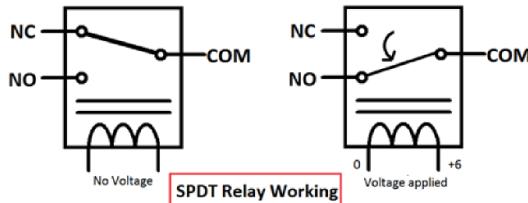


Figura 3.20: Diagrama simplificado de un Relé.

Un Relé, como el e la figura 3.22, es un dispositivo que permite accionar magnéticamente un interruptor interno, en donde la principal característica es que tanto la bobina que produce el campo magnético como el interruptor, pueden pertenecer a circuitos distintos, es decir, podríamos accionar la bobina del dispositivo como si se tratara del motor DC de nuestro ejemplo anterior y el interruptor utilizarlo para accionar una ampolleta de nuestra casa. Lo anterior, además otorgar una *Aislación Galvánica* entre nuestra Arduino y la red de 220VAC, es decir, no existe un camino eléctrico conductor entre ambas partes del circuito.

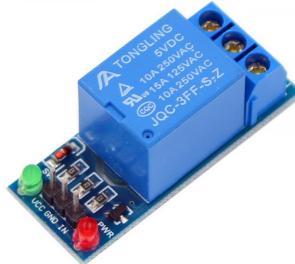
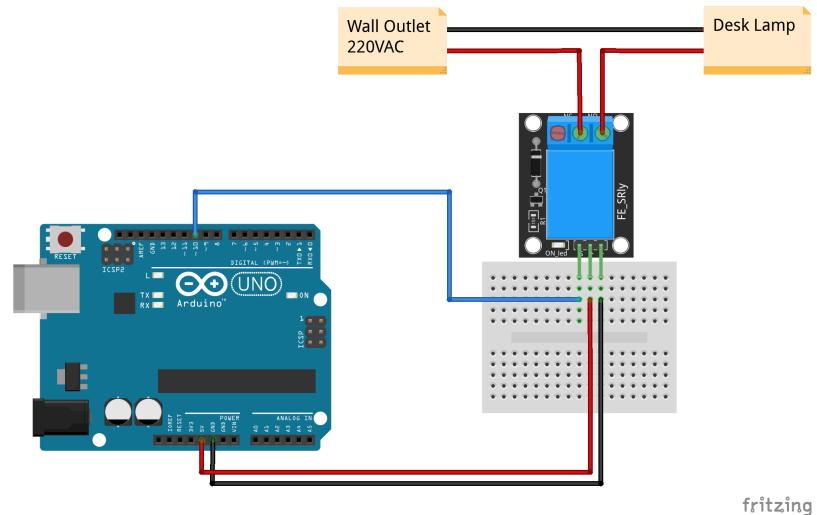


Figura 3.21: Módulo Relé Integrado.

En la figura 3.21, se muestra un Módulo Relé integrado, que permite manipular cargas de 220VAC directamente desde algún pin de nuestra Arduino.

3.7.1. Actividad

- Realice la conexión del sensor Ping (véase la sección 3.3) y la conexión del Módulo Relé a los pines que usted estime conveniente. Para la parte de potencia del circuito, puede utilizar como guía la siguiente imagen.
- Smart-Home:** Desarrolle un programa, de manera que la luz se encienda, en cuanto alguien se acerque a menos de 1 metro del sensor ultrasónico y que esta permanezca encendido durante 5 segundos.



fritzing

Figura 3.22: Circuito con Relé y carga de 220VAC.

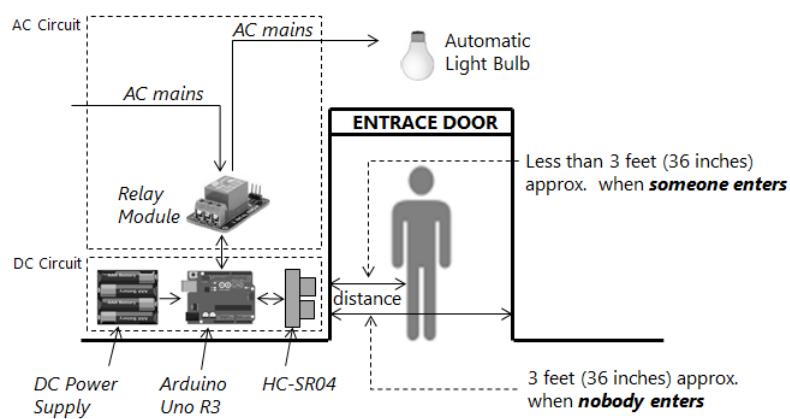


Figura 3.23: Ejemplo de Aplicación Smart-Home.

Capítulo 4

Introducción a los Sistemas de Control Automático

4.1. Automatización

En general, cuando se habla de *Automatización* inmediatamente se asocia el concepto a sistemas inteligentes, sistemas que realizan tareas repetida e incansablemente o sistemas que controlan alguna variable de interés, como por ejemplo puertas de recintos comerciales que se abren cuando algún cliente se acerca a ellas, rápidas líneas de ensamblaje que completan varias tareas al minuto o un climatizador que mantiene la temperatura que el usuario quiera independiente de las condiciones ambientales exteriores; son todos sistemas automáticos que en cierta medida no requieren la intervención de un ser humano para su pleno funcionamiento.

Automático

Que funciona por sí solo o que realiza total o parcialmente un proceso sin ayuda humana.

Es posible diferenciar dos tipos de sistemas que tienen propósitos diferentes pero no dejan de ser automáticos. Por ejemplo

- Se tiene un estanque con dos botones que al ser presionados comanda el vaciado o llenado completo del recipiente.
- Se tiene un sistema que controla el nivel del estanque que es capaz de purgar los excesos de nivel perdido mediante la apertura gradual de una válvula.

La diferencia fundamental entre estos dos sistemas, es que uno automatiza una *tarea* mientras que el otro realiza una *regulación* automática permanente en el tiempo.

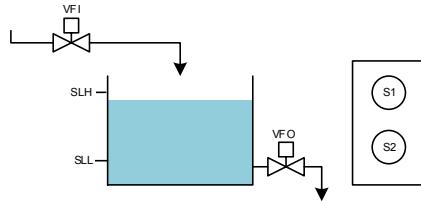


Figura 4.1: Estanque de Control Discreto del Nivel.

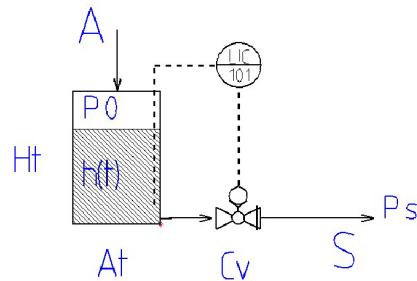


Figura 4.2: Diagrama P& ID del Control Continuo del Nivel de Estanque

4.2. Control de Eventos Discretos

Es la automatización de tareas repetitivas y eventos discretos que no requieren mayor inteligencia, por lo que pueden ser ejecutados por máquinas de forma reiterativa y a gran velocidad. Se dividen en tareas y eventos que tienen un comienzo y un final. Dentro de este tipo de sistemas se encuentran.

- Partidas directas e inversores de giro con elementos de comando eléctrico.
- Encender o apagar las luces de una vivienda a horarios determinados.
- Encender o apagar las luminarias en la vía pública cuando se ha escondido el sol.
- Ensamblar piezas mediante brazos robóticos.

4.3. Control Regulatorio

Consiste en mantener variables físicas en un valor deseado según estrategias acuñadas por la *Teoría de Control* que generalmente requieren un análisis matemático avanzado de la física que involucra el proceso que se quiere controlar.

En su forma más básica, el control regulatorio se logra siguiendo la *Estructura General de Control* detallada a continuación.

4.4. Estructura general de control

Al representar las distintas partes del sistema como bloques las partes necesarias para controlar una variable junto con las interconexiones entre las mismas son las siguientes.

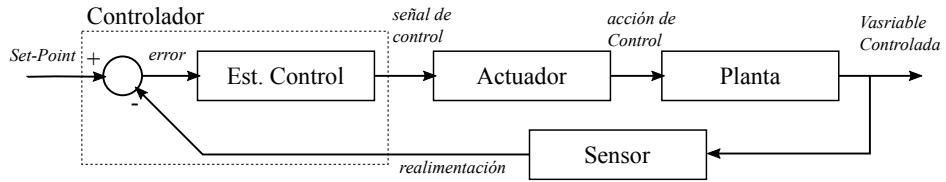


Figura 4.3: Estructura General de un Sistema de Control.

4.4.1. Bloques del Sistema

- **Controlador:** Es un equipo al que se ingresa el valor de set point, se conecta el sensor y el actuador. Contiene la Estrategia de Control y en base a esta comanda al actuador según la magnitud de la diferencia entre el valor deseado y el valor actual que mide el sensor.
- **Actuador:** Es el encargado de ejercer la *Acción de Control*, es comandado por el controlador y es el que provee el esfuerzo necesario para cambiar el estado actual de la variable de salida del proceso. Se conoce tambien con el nombre de *Elemento Final de Control*.
- **Proceso:** Comunmente se le llama *Planta*, y es el proceso físico que se quiere controlar, por ejemplo un estanque con su dinámica de llenado y vaciado, una caldera con toda la física involucrada en el calentamiento del agua y su transformación a vapor. Este bloque, recibe la acción de control cambiando su estado, siendo este medido por el sensor e informando al controlador.
- **Sensor:** Se encarga de informar al controlador acerca del estado actual del proceso que se quiere controlar, es el encargado de producir la *Realimentación* necesaria para que el sistema sea controlado.

4.4.2. Señales del Sistema

Son las líneas que interconectan cada uno de los bloques.

- **Set-Point SP:** Es el valor deseado para el sistema en cuestión, tiene la misma magnitud que la variable de salida, por ejemplo si se controla el nivel de un estanque, el set point, será un nivel determinado, por ejemplo 5m de altura. También se conoce con los nombres de *Señal de Consigna*, *Señal de Referencia*, *Entrada del Sistema*, entre otros.
- **Error:** Es el valor resultante de la resta del Set-point con el Valor actual del proceso informado por realimentación que provee el sensor. Cuando este valor es cero se dice que el sistema esta controlado.
- **Señal de Control MV:** Es la señal de comando para el actuador generada en el controlador de acuerdo al error calculado segúin lo que dicte la estrategia de control.
- **Acción de Control:** Es el esfuerzo físico que ejerce el actuador, puede ser cerrar una válvula, encender un calefactor, encender un ventilador, etc.

- **Realimentación PV:** Es la señal eléctrica enviada por el sensor para informar al controlador el estado actual de la planta.

4.5. Aplicación de Arduino como Sistema de Control Regulatorio

A continuación, se presenta un sistema de control de un proceso físico, en donde Arduino actúa como controlador. El sistema consiste en la levitación de una bolita de poliestireno a una altura deseada, por medio del empuje de una corriente de aire producida por un ventilador de CPU; la altura es sensada de forma continua por un sensor ultrasónico Ping y el Set-Point se modifica mediante un potenciómetro conectado al Arduino. En la figura 4.4, se muestra un esquema del sistema.

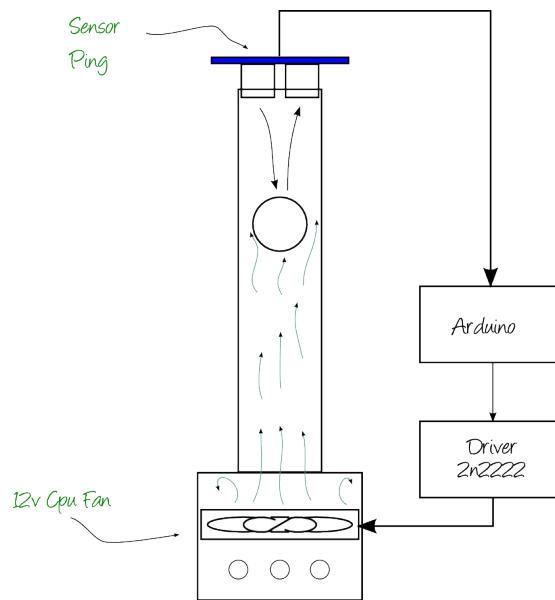


Figura 4.4: Sistema de control de levitación por aire.

A grandes rasgos, el sistema permite manipular la altura a la que levita la bolita, mediante un potenciómetro. Si analizamos el sistema mediante la *Estructura General de Control* de la figura 4.3, tenemos el caso mostrado a continuación. (figura 4.5)

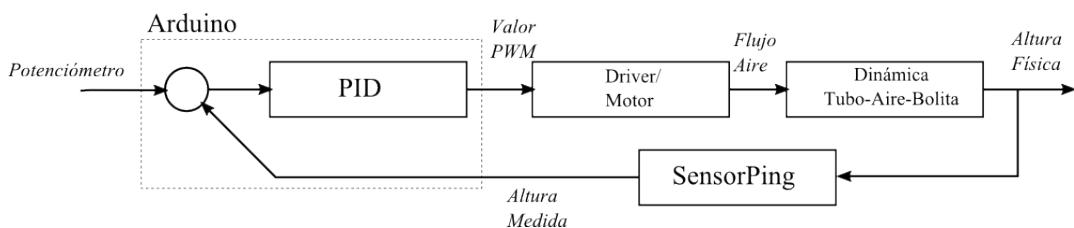


Figura 4.5: Estructura general de control aplicada al sistema.

Cabe notar, que en el bloque correspondiente a la *Estrategia de Control*, encontramos las siglas PID, puesto que es precisamente el nombre de la estrategia que se utilizará para calcular la potencia que se debe enviar al actuador para obtener una altura determinada.

4.5.1. Control Proporcional Integral y Derivativo PID

Es una estrategia de control, que permite calcular la magnitud de la *Acción de Control* para que el valor de la *Realimentación* se iguale al valor del *Set-Point* y que aunque el cálculo no lleve a un buen resultado, el sistema por sus característica iterativa logrará llegar al punto buscado.

El control PID se basa en tres constantes, llamadas **Proporcional, Integral y Derivativa**, las cuales son ponderaciones que se otorgan al valor de la acción de control en base al *Error*, es decir, podemos tener un sistema sensible a los tres siguiente parámetros:

- **Constante Proporcional:** permite ajustar la sensibilidad del sistema al tamaño del error, es decir, mientras mayor sea el error, mayor será la acción que se ejerza sobre el sistema.
- **Constante Proporcional:** permite ajustar la sensibilidad del sistema al tiempo que este aguantará que se presente el error, es decir, a mayor constante integral, el sistema tolerará durante menos tiempo un error estático aumentando el valor de la acción de control en el tiempo.
- **Constante Proporcional:** ajusta la sensibilidad del sistema a la tasa de cambio del error, es decir, si el sistema se acerca o se aleja muy rápido del Set-Point, la acción de control aumentará o disminuirá acorde a esa velocidad.

4.5.2. Código que Implementa el Control del Sistema

A continuación, se presenta el código que permite controlar la levitación de la bolita de poliestireno.

```
1 #include <PID_v1.h>
2
3 const int pingPin = 7;                      //Ultrasonic sensor Pin
4 const int fanPin = 8;                        //Actuator Pin (fan)
5 const int potPin = 5;                         //Potentiometer to set the Sp
6
7 double Sp, Pv, Mv;                          //Values for PID control
8
9 //Compensator Values
10 double kp = 5;                             //Proportional
11 double ki = 3;                             //Integral
12 double kd = 3;                             //Derivative
13
14 //Compensator declaration
15 PID myPID(&Pv, &Mv, &Sp, kp, ki, kd, REVERSE);
16
17 void setup() {
18
19   Serial.begin(9600);
20
21   Pv = 29;                                  //initial PV Value to trigger the actuator.
22   Sp = 18;                                  //initial SetPoint
23
24   myPID.SetMode(AUTOMATIC);
25   myPID.SetOutputLimits(0,255);
```

```

26
27 }
28
29 void loop() {
30
31   Sp = map(analogRead(potPin), 0, 1023, 3, 29);
32   Pv = usonicMeas();
33
34   myPID.Compute();
35
36   analogWrite(fanPin, Mv);
37
38   showVariables();           //list Sp,Pv and Mv on Serial Monitor
39
40 }
41
42 long usonicMeas(){
43
44   //shape the pulse
45
46   pinMode(pingPin, OUTPUT);
47   digitalWrite(pingPin, LOW);
48   delayMicroseconds(2);
49   digitalWrite(pingPin, HIGH);
50   delayMicroseconds(5);
51   digitalWrite(pingPin, LOW);
52
53   //Measure Time of Flight
54
55   pinMode(pingPin, INPUT);           //act as input
56   long duration = pulseIn(pingPin, HIGH); //Read the pulse
57   return duration / 29 / 2;         //return value in centimeters
58
59 }
60
61 void showVariables(){
62
63   Serial.print(Sp);
64   Serial.print(", ");
65   Serial.print(Pv);
66   Serial.print(", ");
67   Serial.println(Mv);
68
69 }

```

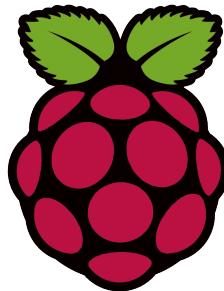
4.5.3. Actividad

En base a la explicación del código anterior, provista por el profesor, realice las siguientes tareas:

1. Identifique a partir del código y la información presentada, el diagrama de conexiones que requiere este circuito, indicando claramente los pines del Arduino donde deben ir conectados los elementos:
 - Sensor Ping.
 - Cpu Fan y driver 2n2222.
 - Potenciómetro.
2. Identifique la sección de código que se encarga de la medición de la altura de la bolita y como funciona esta.
3. Explique que función cumple la instrucción `analogRead` dentro de todo el sistema de control.

Capítulo 5

Internet de las Cosas con Raspberry Pi



Raspberry Pi es un pequeño computador del tamaño de una tarjeta de crédito, el cual podemos conectar fácilmente a una televisión vía HDMI o RCA como salida de video. Además al ser un computador, podemos utilizarla, con un teclado y un mouse. Tiene conexión vía Ethernet, bus USB y pines GPIO (General Purpose Input/Output), para que podamos interactuar con nuestra placa con sensores, botones, entre otros. Pese a que el sistema operativo que se utiliza por defecto como parte del proyecto; **Raspbian**, tiene capacidades de reproducción de vídeo en alta definición, procesadores de texto tipo *Office* y juegos tipo *Arcade*, realmente la mayor utilidad de este sistema computacional, se encuentra en el desarrollo de sistemas computacionales embebidos, es decir, aquellas aplicaciones que requieran de procesamiento de datos, como *Data-Mining*, visión artificial, recolección de Big-Data, entre otros y que no necesiten de un entorno gráfico o siquiera un escritorio, pues el hardware se halla inserto en algún proceso o sistema muy lejos de parecerse el escritorio de una oficina con teclado, mouse y pantalla.

5.1. Primer Encendido y Configuración Para Uso Remoto

Este curso, considera el uso de la variante de Raspberry Pi 3, precargada con el sistema operativo Raspbian, entre las especificaciones computacionales de esta tarjeta se destaca:

- Quad Core 1.2GHz Broadcom BCM2837 64bit CPU.



Figura 5.1: Raspberry Pi 3 Model B.

- 1GB RAM.
- Conectividad wireless LAN y Bluetooth de bajo consumo energético.
- 40 pines GPIO.
- 4 puertos USB 2.0.

Nuestro primer encendido, lo haremos como si se tratase de un computador de escritorio, por lo tanto necesitamos contar con un **Teclado, un Mouse y una Pantalla**, además de una **tarjeta SD previamente cargada con el sistema operativo Raspbian** y respectiva alimentación de 5 volts, capaz de proveer 2.5A. Con ello, podremos a punto nuestra Raspberry para poder deshacernos de todo el aparataje que conllevan los periféricos nombrados, pudiendo acceder desde la comodidad de nuestro computador portatil por medio de una conexión de **Escritorio Remoto VNC o SSH** en caso de no ser necesario un entorno gráfico.

5.1.1. Actividad: Configuración de Raspberry para conexión remota

1. Conectamos Teclado, Mouse, Monitor, Tarjeta SD y Alimentación a nuestra Raspberry. Aquí deberíamos ver un encendido típico de una maquina Linux, primero iniciando y luego detenerse en el escritorio clásico de Gnome. (figura 5.2)
2. Con el fin de habilitar el acceso remoto a nuestra Rpi, vamos al *Applications Menu>Preferences>Raspberry Pi Configuration*, una vez abierta la ventana correspondiente, seleccionamos la pestaña *Interfaces* y marcamos las opciones que nos interesen. Con fines de desarrollo, es común habilitarlas todas. (figura 5.3)
3. Instalamos el software que nos permitirá acceder de manera gráfica nuestro escritorio **Real VNC Server** mediante el clásico terminal de linux y el comando Apt.

```
sudo apt-get install realvnc-vnc-server
```

Curiosamente, esta distro de linux, no pide contraseña de administrador al ejecutar el comando sudo. De todas maneras, el usuario del escritorio es Pi y la contraseña de este es raspberry, y en caso del usuario root, la contraseña es la misma.

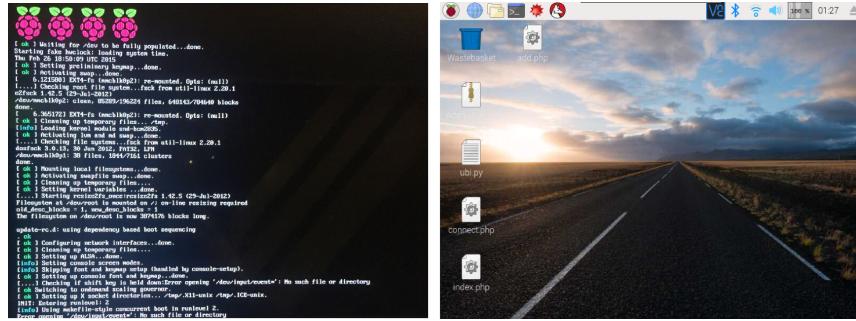


Figura 5.2: Pantallas de inicio al encender Raspberry Pi.

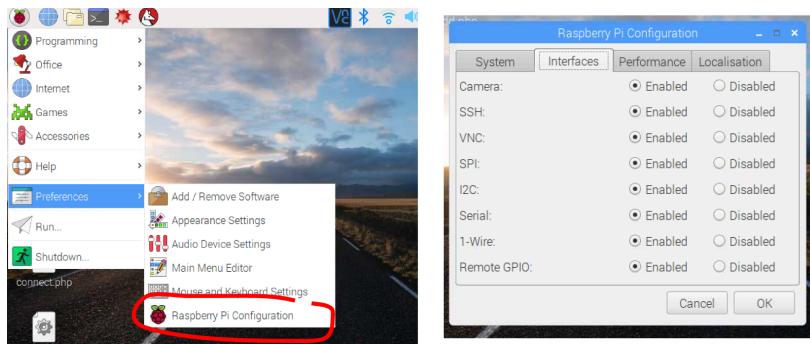


Figura 5.3: Configuración de interfaces de Raspberry.

4. Conectamos nuestra Rpi a la red. Ya sea por medio de WiFi o de manera cableada y registramos la IP que nos ha asignado el Router, para poder acceder con ella nuestro escritorio remoto.

En el terminal ingresamos `sudo ifconfig`, ante lo cual nos debería aparecer una respuesta del tipo mostrado en la figura 5.4. Mientras no desconectemos nuestra Raspberry, podremos acceder mediante esta IP desde nuestro PC o portátil. En caso de desconectarnos, aun existen muchas maneras de descubrir o incluso fijar la IP de nuestra Raspberry, ello escapa a los contenidos de este curso, pero un buen manejo técnico de redes de datos siempre podrá resolverlo.

5. Finalmente, desde nuestro portátil o PC podremos ingresar de manera remota a nuestra Rpi con la IP, usuario y contraseña correspondientes, desde el software RealVNC, tal como se muestra en la imagen 5.5.

```

pi@raspberrypi:~ $ sudo ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536
    inet 127.0.0.1 netmask 255.0.0.0
        inet6 ::1 prefixlen 128 scopeid 0x10<host>
            loop txqueuelen 1 (Local Loopback)
            RX packets 25 bytes 1484 (1.4 KiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 25 bytes 1484 (1.4 KiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

wlan0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST> mtu 1500
    inet 192.168.1.13 netmask 255.255.255.0 broadcast 192.168.1.255
        inet6 fe80::26cd:f45e:fd09:83f prefixlen 64 scopeid 0x20<link>
            ether b8:27:eb:f2:c6:b2 txqueuelen 1000 (Ethernet)
            RX packets 5585 bytes 388471 (379.3 KiB)
            RX errors 0 dropped 0 overruns 0 frame 0
            TX packets 6448 bytes 2831194 (2.7 MiB)
            TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0

pi@raspberrypi:~ $ 

```

Figura 5.4: Respuesta del comando sudo ifconfig.

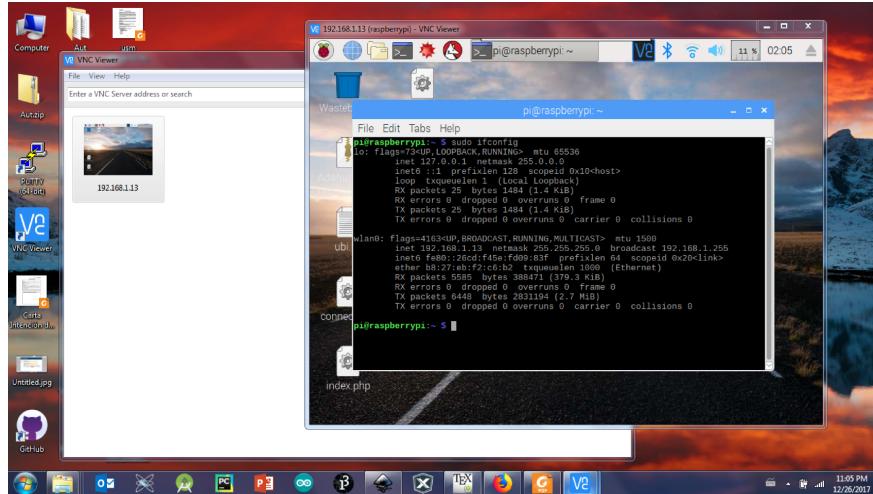


Figura 5.5: Conexión VNC con Raspberry desde Windows 7.

5.2. Control de GPIO's en Python

Como ya se vio antes, en la introducción de este curso, los GPIO's son pines de entradas o salidas de propósito general, en términos simples, son pines digitales como los de Arduino que pueden ser programados por medio de lenguajes de alto nivel como **Python**.

Dependiendo del modelo de Raspberry a utilizar, puede que este traiga o no escritos los nombres de sus respectivos pines, como por ejemplo, el caso de Pi Zero que no trae nombre en sus pines. De todas formas los modelos siguen el mismo *Pinout* con pequeñas variaciones, por ejemplo en la figura 5.6 se muestra el correspondiente al modelo B+.

Nota: La mayoría de los GPIO's tienen conexión directa con el chip Broadcom, por lo tanto al lidiar con

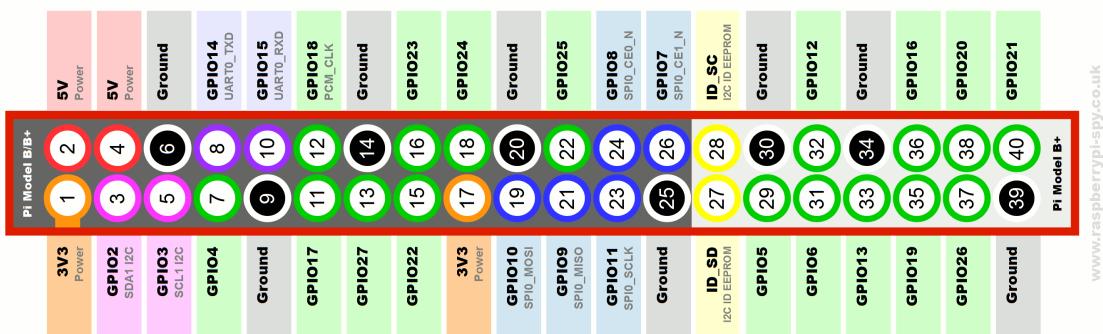


Figura 5.6: Enumeración de GPIO's de Raspberry Pi modelo B+. (tomada de [9])

circuitos conectados a estos pines, existe un gran riesgo de inhabilitar nuestra Raspberry.

5.2.1. Actividad: Manejo de GPIO's desde Python

- Preparar el circuito a utilizar durante todo este capítulo, para ello realizar el montaje mostrado en figura 5.7.

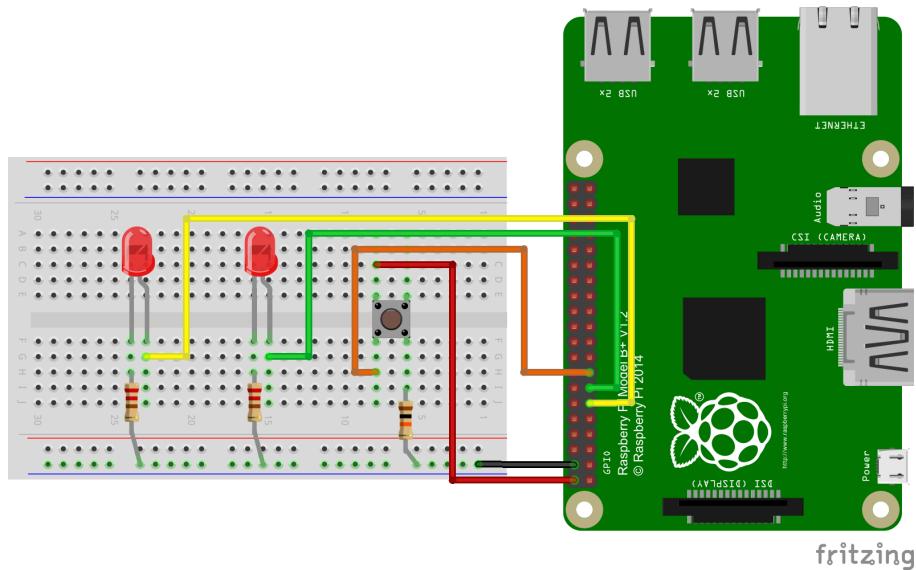


Figura 5.7: Circuito de prueba para control de GPIO's desde Python.

- Instalar o revisar la instalación del interprete de Python 3 y el módulo Rpi.GPIO mediante el siguiente comando desde una terminal.

```
sudo apt-get install python-dev python-rpi.gpio
```

Lo anterior, procederá a instalar lo mencionado, o en su defecto, nos permitirá verificar una instalación previa.

3. Por medio de un Terminal, creamos un archivo para nuestro código, con el comando:

```
nano blink.py
```

Luego, en el editor de texto, escribimos el siguiente código:

```
1 import RPi.GPIO as GPIO
2 import time
3 GPIO.setmode(GPIO.BCM)
4 GPIO.setup(17, GPIO.OUT) ## GPIO 17 como salida
5 GPIO.setup(27, GPIO.OUT) ## GPIO 27 como salida
6
7 def blink():
8     print "Comienzo Programa..."
9     while 1:
10         GPIO.output(17, True)    ## Enciendo el 17
11         GPIO.output(27, False)   ## Apago el 27
12         time.sleep(1)          ## Esperamos 1 segundo
13         GPIO.output(17, False)   ## Apago el 17
14         GPIO.output(27, True)    ## Enciendo el 27
15         time.sleep(1)          ## Esperamos 1 segundo
16     GPIO.cleanup() ## Hago una limpieza de los GPIO
17
18 blink() ## llamamos a la funcion
```

Una vez escrito el código, presionamos CTRL+X y aceptamos guardar los cambios.

4. Finalmente, ejecutamos nuestro código con permisos de administrador, mediante la siguiente instrucción:

```
sudo python blink.py
```

Si todo ha resultado bien, hasta esta parte deberíamos ver en el terminal el mensaje Comienzo Programa... y nuestros leds deberían comenzar a parpadear según los tiempos programados.

5. Finalmente, modifique los tiempos de encendido de los Led y verifique el funcionamiento de la implementación.

5.3. Servicio IoT Ubidots.com



Ubidots.com es un servicio para aplicaciones de *Internet de las Cosas IoT*, que existe desde 2014, presta servicios comerciales de almacenamiento de datos en la nube y posee un muy buen soporte, independiente si se trata de cuentas de pago o educacionales, además de contar con una creciente comunidad que genera material para la mayoría de las plataformas hardware y software actuales, como Raspberry Pi, Arduino, NodeMCU, entre otras.

Su funcionamiento es bastante simple. Se basa en cuentas de usuario, aseguradas por un *Token*¹, en donde uno puede crear dispositivos, los que a su vez pueden contener variables con identificadores únicos, las cuales, contienen el historial de datos almacenados que bien podrían ser enviados por elementos tanto físicos como elementos software. Una vez puestas a punto, las variables y dispositivos, se pueden crear vistosos *Dashboards* que permiten tanto visualizar como manipular los datos contenidos en las variables. (figura 5.8)

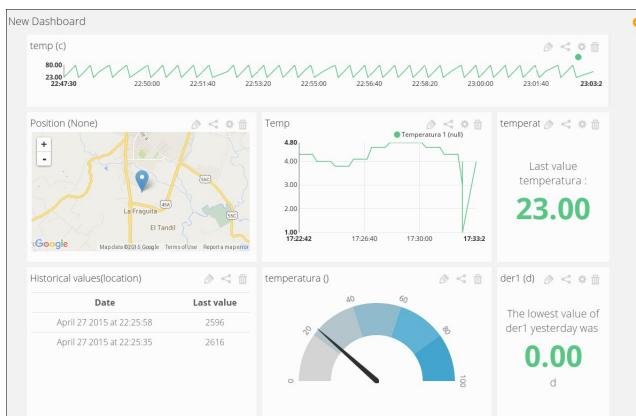


Figura 5.8: Ejemplo de Dashboard en Ubidots.

Ubidots cuenta con API's y librerías, que permiten tanto subir como bajar datos, desde todo tipo de aplicaciones, incluyendo un módulo para Python.

5.3.1. Actividad: Cuenta y Dispositivos en Ubidots

1. Diríjase al sitio ubidots.com y en el botón *Signup*, complete la información requerida y siga los pasos para crear una cuenta, necesitará de una dirección de correo electrónico.
2. En la parte superior de su cuenta, en *Devices*, cree un nuevo dispositivo y llámelo *Raspberry* o el nombre que usted prefiera, a su vez, dentro del dispositivo cree las variables *cuenta*, *led1* y *led2*. Debería ver pantallas como las que se muestran en la figura 5.9.

¹Identificador único para cada cuenta

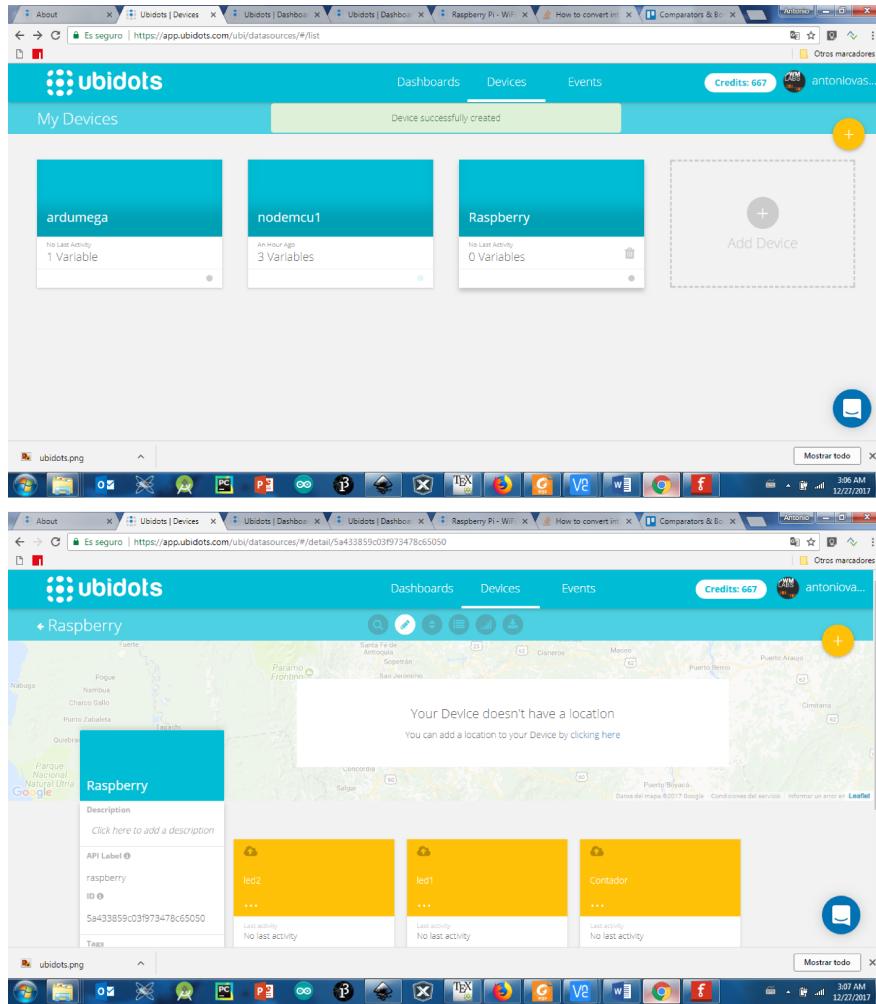


Figura 5.9: Creación de un dispositivo y variables.

- Finalmente, tome nota de su *Token* y sus *Variable ID's*, cómo se muestra en la figura 5.10. Se recomienda seguir las instrucciones del relator.

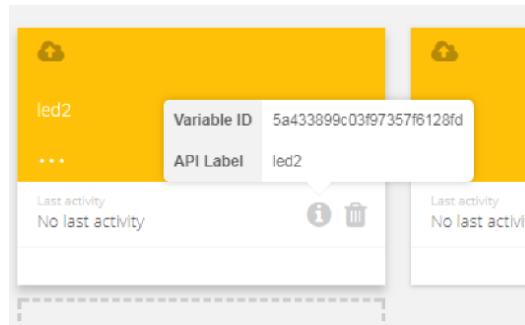


Figura 5.10: Identificador de variable.

4. En la parte superior de su cuenta, en *Dashboard*, cree una pantalla escogiendo *widgets* de manera que se muestre algo como lo mostrado en la figura 5.11.

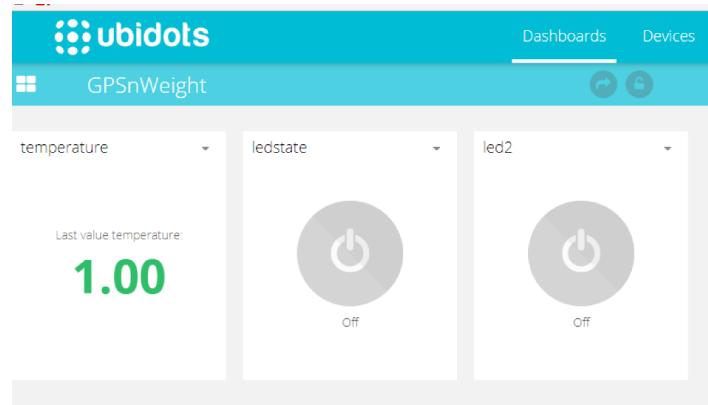


Figura 5.11: Dashboard con nuestras variables.

5.3.2. Actividad: Envío de datos a Ubidots

A continuación, se presenta una implementación que genera una variable en un script en Python y envía dicha variable a una cuenta en Ubidots. Lo anterior, puede ser fácilmente modificado para el envío de datos tomados por algún sensor.

1. Primero, necesitamos instalar el módulo que permite que Python comunique con Ubidots. Para ello en un terminal ejecutamos:

```
$ sudo apt-get install python-setuptools  
$ sudo easy_install pip  
$ sudo pip install ubidots
```

2. Manteniendo el circuito de la figura 5.7, de actividades anteriores, creamos un archivo llamado `contador.py` con el siguiente código:

```
1 import RPi.GPIO as GPIO  
2 from ubidots import ApiClient  
3 import time  
4 GPIO.setmode(GPIO.BCM)  
5 GPIO.setup(22, GPIO.IN)  
6 GPIO.setup(17, GPIO.OUT)  
7  
8 api = ApiClient(token='SU-TOKEN')  
9 contador = api.get_variable('ID-VARIABLE-CONTADOR')  
10  
11 def program():  
12     print "Ejecucion iniciada..."  
13     i = 0  
14     while 1:
```

```

15         if GPIO.input(22):
16
17             while GPIO.input(22):
18                 pass
19                 i=i+1
20                 print i
21                 contador.save_value({"value": i})
22             GPIO.cleanup()
23
24 program() ## Hago la llamada a la funcion

```

El código resulta bastante intuitivo. En síntesis, cada vez que se presiona el botón conectado a GPIO22 aumenta un valor de cuenta en una unidad y envía ese valor la variable *Contador* de nuestra cuenta.

- Finalmente, ejecute el código y verifique su funcionamiento; al presionar el botón conectado a la Raspberry el valor de cuenta, debería aumentar al mismo tiempo que lo hace en el Dashboard.

5.3.3. Actividad: Accionamiento de dispositivos desde Ubidots

A continuación, se presenta una implementación, que permite accionar elementos discretos desde una cuenta en Ubidots. Lo anterior, mediante el uso de módulos relé puede perfectamente permitir la activación o desactivación de artefactos electrodomésticos desde la nube.

- Manteniendo el circuito de la figura 5.7, de actividades anteriores, creamos un archivo llamado `leds.py` con el siguiente código:

```

1 import RPi.GPIO as GPIO
2 from ubidots import ApiClient
3 import time
4 GPIO.setmode(GPIO.BCM)
5 GPIO.setup(27, GPIO.OUT)
6 GPIO.setup(17, GPIO.OUT)
7
8 api = ApiClient(token='SU-TOKEN')
9 led2 = api.get_variable('VAR-ID-LED2')
10 led1 = api.get_variable('VAR-ID-LED1')
11
12 def program():
13     print "Ejecucion iniciada..."
14
15     while 1:
16         valorLed1 = led1.get_values(1)
17         valorLed2 = led2.get_values(1)
18
19         print "led1" + str(valorLed1[0]['value'])+", led2"
20         + str(valorLed2[0]['value'])
21
22         if valorLed1[0]['value']==1.0:
23             GPIO.output(17,True)

```

```

23         else:
24             GPIO.output(17, False)
25
26         if valorLed2[0]['value'] == 1.0:
27             GPIO.output(27, True)
28         else:
29             GPIO.output(17, False)
30
31         if valorLed2[0]['value'] == 1.0:
32             GPIO.output(27, True)
33         else:
34             GPIO.output(27, False)
35     GPIO.cleanup()
36
37 program() ## Hago la llamada a la funcion

```

En este caso, el código, se encarga de leer el valor de las variables *Led1* y *Led2* en nuestra cuenta de Ubidots y en base al valor que estos presenten, enciende o apaga los GPIOS 27 y 17.

2. Ejecute el código y verifique su funcionamiento. El terminal de linux debería permanecer mostrando los valores 0.0 ó 1.0 para los leds 1 y 2, mientras que al presionar los botones del dashboard mostrado en la figura 5.11, los leds conectados a nuestra Raspberry deberían cambiar su estado.
3. Adicionalmente, puede descargar la aplicación *Ubidots Explorer* para smartphones con sistema operativo Android y comandar su aplicación desde allí. Tal como se muestra en la figura 5.12.

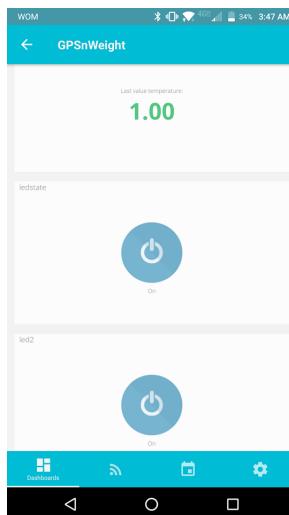


Figura 5.12: App Ubidots Explorer.

Bibliografía

- [1] Dogan Ibrahim, *Programación de Microcontroladores Pic*, Editorial Marcomobo, 2007.
- [2] Hoja de Datos ATmega328p, *8-bit AVR Microcontrollers ATmega328P DATASHEET COMPLETE*
[http://www.atmel.com/Images/
Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf](http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf)
- [3] Introducción a Arduino, *What is Arduino?*
<https://www.arduino.cc/en/Guide/Introduction>
- [4] Adafruit Industries, *SparkFun Inventor's Guide*
Guía ilustrada adjunta al kit de iniciacion en Arduino Sparkfun Inventors Kit.
- [5] Serial – Arduino Reference
<https://www.arduino.cc/reference/en/language/functions/communication/serial/>
- [6] Texas Instruments, *LM35 Temperature Sensor Datasheet*
<http://www.ti.com/lit/ds/symlink/lm35.pdf>
- [7] Antonio David Vásquez Briones, *Conceptos de Automatización*
Apunte para asignatura Sistemas de Control Automático, Inacap Concepción.
- [8] Sitio de Descargas de Real VNC para disintos SO's.
<https://www.realvnc.com/es/connect/download/vnc/>
- [9] Guía para GPIO's en Raspberry Pi.
Simple Guide to the RPi GPIO Header and Pins
<https://www.raspberrypi-spy.co.uk/2012/06/simple-guide-to-the-rpi-gpio-header-and-pins>
- [10] Documentación Ubidots para Raspberry Pi
Raspberry Pi - WiFi/Ethernet
<https://ubidots.com/docs/devices/raspberrypi.html#raspberry-pi-wifi-ethernet>

Agradecimientos

Se agradece a las siguientes personas que contribuyeron de manera directa o indirecta en la realización de este curso.

- **Marcos Abarzúa Fuentealba** – Asesor Área Informática y Telecomunicaciones
- **Eduardo Diaz Manríquez** – Coordinador de Especialidad Área Electricidad y Electrónica Inacap Concepción
- **Pamela Avilés Belmar** – Coordinadora de Especialidad Área Informática y Telecomunicaciones Inacap Concepción
- **Juan Carlos Spichiger Stuardo** – Director de Carreras Informática, Telecomunicaciones y Diseño Inacap Concepción
- **Rodrigo Ferreira Soto** – Coordinador Área Electricidad y Electrónica Inacap Concepción

***Todo el material del curso, ha sido redactado, desarrollado y probado, íntegramente por su autor. En caso de incluirse información de terceros, esta se encuentra debidamente referenciada.*

Los códigos fuente expuestos e incluso el código fuente del mismo documento, están disponibles en el repositorio Github <https://github.com/AnthonyMake/capacitacionArduinoRaspberry>.

Antonio David Vásquez Briones
Ingeniero Civil en Automatización
Licenciado en Ciencias de la Ingeniería
Académico Área Electricidad y Electrónica Inacap Concepción