INSTITUTO TECNOLOGICO DE TUXLA GUTIERREZ

INGENIERIA ELECTRICA

REPORTE DE RESIDENCIA

INTERFACES DE RED PARA MONITOREO DE SEÑALES ELÉCTRICAS Y CONTROL DE DISPOSITIVOS BASADO EN PYTHON-GLADE

ASESOR

DR. RUBEN HERRERA GALICIA

ALUMNO
JOSÉ ANTONIO MORALES SÁNTIZ

TUXTLA GUTIERREZ, CHIAPAS, 18 DE DICIEMBRE 2014

Índice Pág.

1. Introducción

1.1 Antecedentes

Los problemas de monitoreo de señales eléctricas están presentes cuando se trata de lograr una exactitud en ellos; de igual forma en el control de los dispositivos, siempre a existido fallos para poder tener un control correcto, el problema surge nuevamente en buscar la manera correcta para monitorear y controlar los dispositivos, conocer el hardware y el software correcto para lograr un mejor resultado.

Los interfaces de red para monitoreo de señales, ha ido evolucionando en las diferentes áreas de la tecnología, sin embargo las señales eléctricas se deben de monitorear con exactitud para evitar daños en los diferentes equipos, de igual forma el control de los dispositivos se ha ido mejorando, con el avance de los nuevos equipos y software que las diferentes compañías y empresas han creado

Existen software que permiten crear interfaces para monitorear las señales y controlar dispositivos, pero estas siempre tienen ciertas limitaciones, ya que solo pueden emplearse en una o dos plataformas y esto hace que se realice un gasto mayor por conseguir diferentes software para cada plataforma, también requeriría de una programación diferente por cada plataforma.

1.2 Estado del Arte

Uno de los primeros ejemplos de mando a distancia fue desarrollado en 1893 por Nikola Tesla y descrito en su patente número 613809, titulado Método de un aparato para el mecanismo de control de vehículo o vehículos en movimiento.

En 1903, Leonardo Torres Quevedo presentó el *telekino* en la Academia de Ciencias de París, acompañado de una memoria y haciendo una demostración experimental. El *telekino* consistía en un autómata que ejecutaba órdenes transmitidas mediante ondas hertzianas; constituyó el primer aparato de radiodirección del mundo, y fue un pionero en el campo del mando a distancia.

En 1956, Robert Adler desarrollo el ―*Zenith Space Command*‖ (*Mando del espacio cenit*), un control sin cables. Era mecánico y usaba ultrasonidos para cambiar el canal y el volumen. Cuando el usuario pulsaba un botón del mando a distancia, hacía un chasquido, de ahí el término ―clicker‖ (*chasqueador*). Cada barra emitía una frecuencia diferente y los circuitos detectaban el ruido.

En la planta Peldar Envigado se implementó un sistema Scada de comunicación que permitirá leer los insumos energéticos de las máquinas a través del software RSEnergyMetrix y que permitirá a su vez visualizar gráficas de consumos, de costos y reportes que muestren variables específicas. A continuación se mencionaran y describirán las etapas que fueron desarrolladas en este proyecto.

La red de monitoreo del INIFAP cuenta con 36 estaciones meteorológicas distribuidas en el estado de Zacatecas. Estas redes registran mediciones en tiempo real arrojando promedios cada 15 minutos de la temperatura, la humedad relativa, la humedad del follaje, la precipitación, la velocidad del viento, la radiación solar y la dirección del viento, los datos son transmitidos vía RF a una distancia máxima de 10 km a la base central en el Campo Experimental Zacatecas.

La red de monitoreo del SMN cuenta con dos estaciones meteorológicas automáticas, una ubicada en el municipio de Guadalupe, Zacatecas y otra en el municipio de Sombrerete, Zacatecas. Dichas estaciones están equipadas con sensores que permiten medir la presión atmosférica, la temperatura, la humedad relativa, la radiación solar, la precipitación, la velocidad y dirección del viento.

Mohsenin Tinoosh (2004), diseñó una interfaz para una tarjeta de red Ethernet/PCI (por sus siglas en inglés *Peripheral Component Interconnect*) usando un FPGA. Desarrollando solo la descripción para enviar y recibir paquetes por medio de una capa física PHY (*Physical Layer*), y una capa de control de acceso al medio MAC (por sus siglas en inglés *Media Access Control*) comerciales.

Bernspang Johan, (2004), desarrolló una interfaz basada en FPGA usando un controlador Ethernet MAC/PHY externo en el procesador MicroBlaze desarrollado por Xilinx, para este proyecto se adquirió un CS8900A el cual no se logró probar de manera adecuada debido al tiempo de adquisición del controlador.

Toledano Ayala, Manuel (2006), diseñó un sistema de monitoreo remoto para un invernadero por medio tecnología inalámbrica para transmisión de datos adquiridos desde unidades remotas (ubicadas fuera y dentro del invernadero) hacia la unidad base, la cual se encuentra conectada a un ordenador a través del puerto de bus universal en serie, donde se analizan los datos y se generan las gráficas en tiempo real de las variables climatológicas medidas.

Macías Fernández, Ricardo (2007) realizó la implementación de tele-instrumentación con el fin de lograr la interconexión de equipos de muestro sísmico, llamado SISMO1, con una estación remota mediante una red de alta velocidad vía Ethernet, para monitoreo en tiempo real. Donde el equipo SISMO1 entrega una señal serial con el protocolo RS-232 que es convertida mediante una tarjeta diseñada para este proyecto a Ethernet, la cual alberga un transceptor de señales.

Méndez Bautista, David (2009), desarrolló un sistema de comunicación que permite la interacción entre una interfaz basada en un servidor web y un sistema basado en un micro controlador, dicho sistema utiliza el monitoreo vía Ethernet y el conjunto de protocolos para comunicación en Internet, en donde se cuenta con una base de datos para almacenar las acciones realizadas. Todo esto a través de un micro controlador.

Martínez Velázquez, Gabriela (2009), implementó la comunicación en red de un equipo de cómputo que maneja una interfaz sistema integral de control contra

incendios (SICCI) con una de las válvulas de control para aspersores de descarga de auto-tanques, haciendo uso de un controlador lógico programable, en la parte de comunicación y monitoreo se formó una interfaz de control de supervisión y adquisición de datos donde los datos obtenidos por el PLC son enviados a un ordenador y de ahí son transmitidos a un servidor web.

Guzmán Gallegos, Ricardo (2009), desarrolló dos aplicaciones de monitoreo, la primera aplicación fue la implementación del accionamiento de un sistema de riego vía Ethernet con interfaz gráfica en LabView, por medio de un PLC se conectó un ordenador a la red en la que se encontraba configurado el sistema de riego, para establecer las horas de riego, los tiempos de encendido, apagado del sistema y la zona fuera ser regada.

Toledano Ayala, Manuel (2010), desarrolló un sistema de telemetría para la medición remota de largo alcance para estaciones de sensores. El sistema fue probado en una aplicación práctica que consiste en el monitoreo remoto de las variables climatológicas en los altos de Chiapas, nombrando al sistema Meteo UAQ.

Arroyo González, Abdiel y Duran Balderas, Erick Martin (2010), plantearon una propuesta para controlar el pH y la temperatura del proceso de la elaboración de yogurt artesanal utilizando una red Ethernet. La conexión se realizó mediante un PLC y controladores Ethernet para hacer la medición de pH y temperatura, para el control de las propiedades del yogurt además de hacer los análisis requeridos para llevar acabo el correcto desarrollo del proceso.

López Gaudencio, Silvia (2010), desarrolló un sistema para el monitoreo y control a distancia de los parámetros de un interruptor de circuito de distribución de energía eléctrica a través de Internet, realizando su implementación en la tarjeta de desarrollo Easyweb 3, que cuenta con un micro controlador que gobierna una interfaz para conectarse a una red de área local.

1.3 Justificación

Vale la pena hacer este proyecto, ya que los interfaces de red que aquí se presenta tienen como objetivo proporcionar un mejor y exacto monitoreo y control por medio de python – glade ya que es multiplataforma y manejo fácil para todos, gracias a glade el diseño se adapta a lo que el usuario requiere y el costo del hardware es muy accesible.

Es de conocimiento general que los interfaces de red para monitoreo de señales, ha ido evolucionando en las diferentes áreas de la tecnología por ello se aprovecha y se logra nuestro objetivo de tener el mejor control de dispositivos y que sea accesible para todos los usuarios, sin dañar a los equipos a usar ya que todo es atraves de la red.

En la actualidad el monitoreo remoto en tiempo real a largo alcance es parte importante de los procesos industriales, ya que ofrece ventajas como el monitoreo

en uno o varios ordenadores de las condiciones actuales del sistema, modificar parámetros de operación del equipo sin tener que ir hasta el sitio de medición, realizar arranques y paros a distancia, generar gráficos histogramas para su posterior análisis y tener vigilancia constante de un sistema o proceso desde cualquier ubicación con acceso a Internet

Python ha sido parte importante de Google desde el principio, y lo sigue siendo a medida que el sistema crece y evoluciona. Hoy día, docenas de ingenieros de Google usan Python y seguimos buscando gente diestra en este lenguaje.

1.4 Objetivo

Diseñar y construir interfaces de red para monitoreo de señales eléctricas y control de dispositivos basado en Python-Glade.

1.5 Metodología

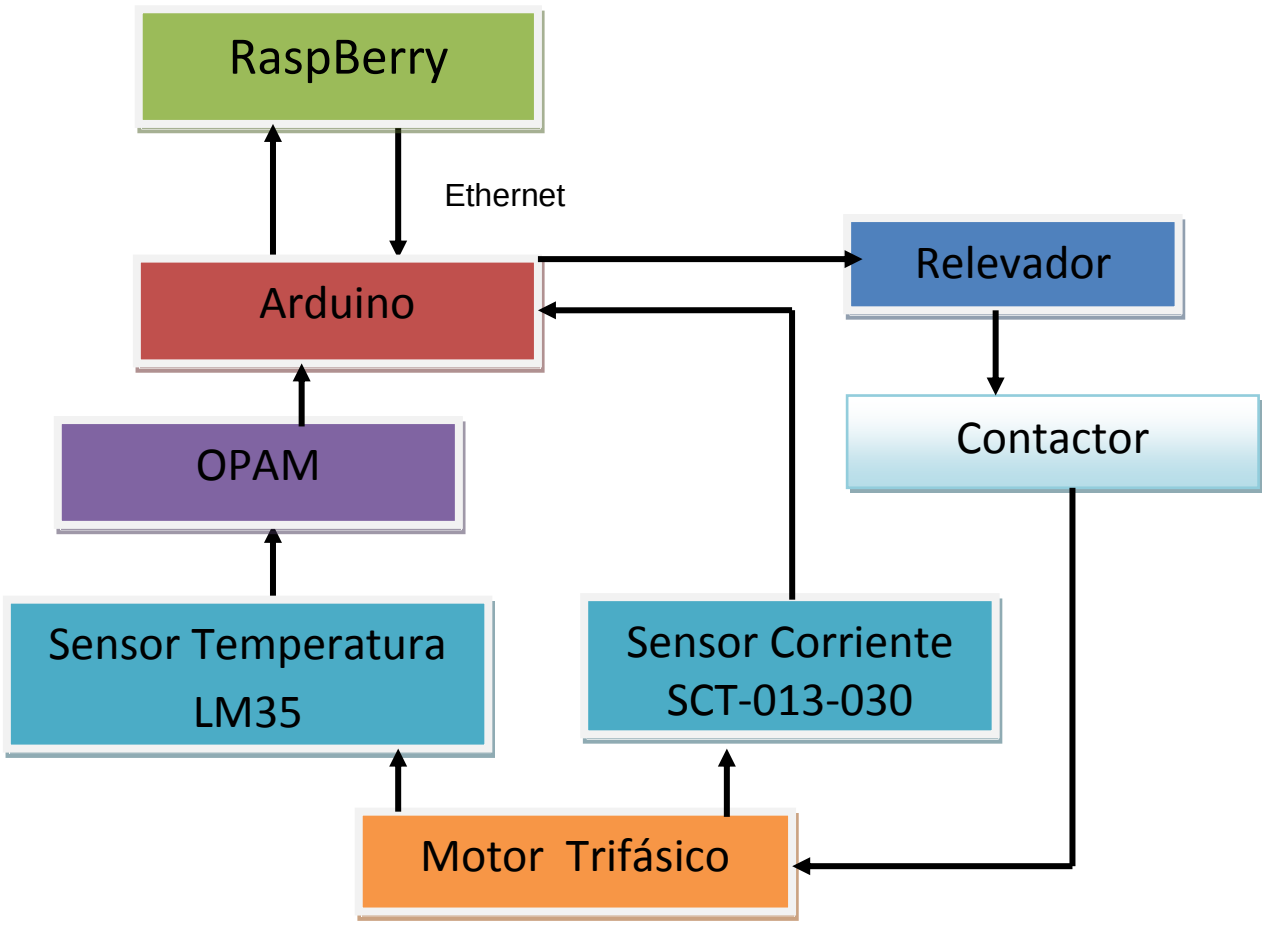


Fig. 1.1 *Diagrama a bloques del hardware.*

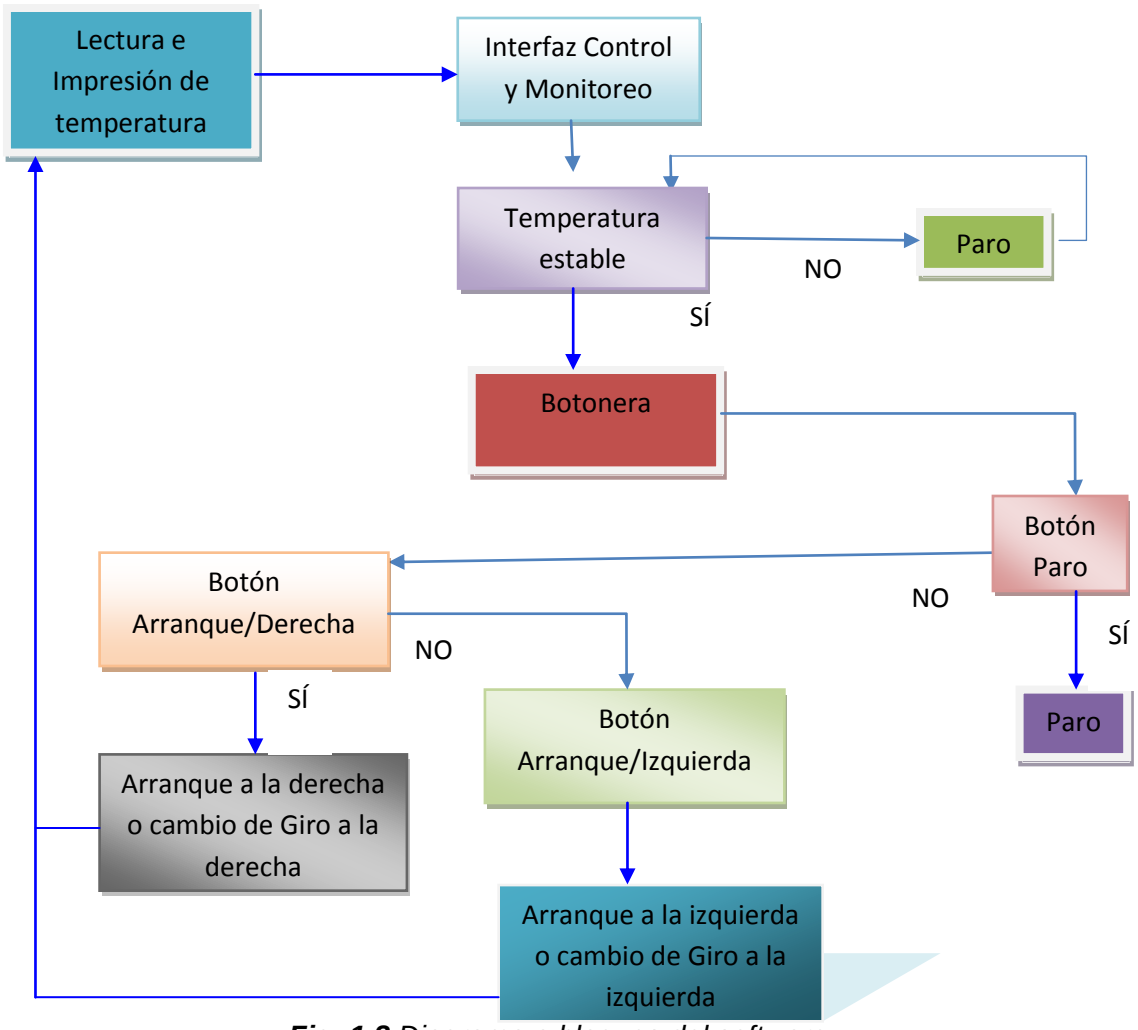


Fig. 1.2 *Diagrama a bloques del software.*

2. Fundamento Teórico

2.1 Python

Python es un lenguaje de programación creado por Guido van Rossum a principios de los años 90 cuyo nombre está inspirado en el grupo de cómicos ingleses “Monty Python”. Es un lenguaje similar a Perl, pero con una sintaxis muy limpia y que favorece un código legible. Se trata de un lenguaje interpretado o de script, con tipado dinámico, fuertemente tipado, multiplataforma y orientado a objetos.

Fig. 2.1 *Logotipo Python.*

Python es fácil de usar, pero es un lenguaje de programación de verdad, ofreciendo mucho mayor estructura y soporte para programas grandes que lo que lo que pueden ofrecer los scripts de Unix o archivos por lotes. Por otro lado, Python ofrece mucho más chequeo de error que C, y siendo un *lenguaje de muy alto nivel*, tiene tipos de datos de alto nivel incorporados como arreglos de tamaño flexible y diccionarios.

Debido a sus tipos de datos más generales Python puede aplicarse a un dominio de problemas mayor que Awk o incluso Perl, y aún así muchas cosas siguen siendo al menos igual de fácil en Python que en esos lenguajes. Python te permite separar tu programa en módulos que pueden rehusarse en otros programas en Python.

Viene con una gran colección de módulos estándar que puedes usar como base de tus programas, o como ejemplos para empezar a aprender a programar en Python. Algunos de estos módulos proveen cosas como entrada/salida a archivos, llamadas al sistema, sockets, e incluso interfaces a sistemas de interfaz gráfica de usuario como Tk.

Un lenguaje interpretado o de script. es aquel que se ejecuta utilizando un programa intermedio llamado intérprete, en lugar de compilar el código a lenguaje máquina que pueda comprender y ejecutar directamente una computadora (lenguajes compilados). La ventaja de los lenguajes compilados es que su ejecución es más rápida. Sin embargo los lenguajes interpretados son más flexibles y más portables.

Python tiene, no obstante, muchas de las características de los lenguajes compilados, por lo que se podría decir que es semi interpretado. En Python, como en Java y muchos otros lenguajes, el código fuente se traduce a un pseudo código máquina intermedio llamado bytecode la primera vez que se ejecuta, generando archivos .pyc o .pyo (bytecode optimizado), que son los que se ejecutarán en sucesivas ocasiones.

Multiplataforma. El intérprete de Python está disponible en multitud de plataformas (UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc.) por lo que si no utilizamos librerías específicas de cada plataforma nuestro programa podrá correr en todos estos sistemas sin grandes cambios.

Orientado a objetos, la orientación a objetos es un paradigma de programación en el que los conceptos del mundo real relevantes para nuestro problema se trasladan a clases y objetos en nuestro programa. La ejecución del programa consiste en una serie de interacciones entre los objetos. Python también permite la programación imperativa, programación funcional y programación orientada a aspectos.

Python es un lenguaje que todo el mundo debería conocer. Su sintaxis simple, clara y sencilla; el tipado dinámico, el gestor de memoria, la gran cantidad de librerías disponibles y la potencia del lenguaje, entre otros, hacen que desarrollar

una aplicación en Python sea sencillo, muy rápido y, lo que es más importante, divertido.

La sintaxis de Python es tan sencilla y cercana al lenguaje natural que los programas elaborados en Python parecen pseudocódigo. Por este motivo se trata además de uno de los mejores lenguajes para comenzar a programar. Python no es adecuado sin embargo para la programación de bajo nivel o para aplicaciones en las que el rendimiento sea crítico.

Algunos casos de éxito en el uso de Python son Google, Yahoo, la NASA, Industrias Light & Magic, y todas las distribuciones Linux, en las que Python cada vez representa un tanto por ciento mayor de los programas disponibles.

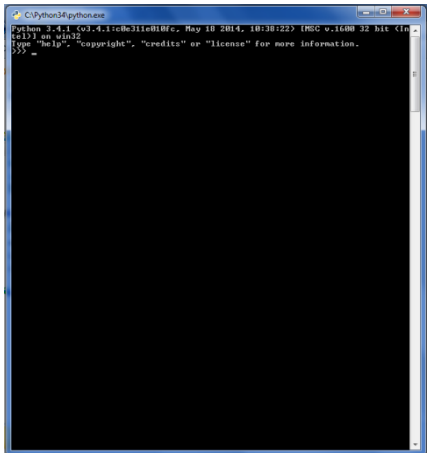


Fig. 2.2 *Ventana de inicio Python.*

Operadores aritméticos. El operador de módulo devuelve el resto de la división entre los dos operandos. La diferencia entre división y división entera no es otra que la que indica su nombre. En la división el resultado que se devuelve es un número real, mientras que en la división entera el resultado que se devuelve es solo la parte entera.

Operador	Descripción	Ejemplo
+	Suma	r = 3 + 2 # r es 5
-	Resta	r = 4 - 7 # r es -3
-	Negación	r = -7 # r es -7
*	Multiplicación	r = 2 * 6 # r es 12
**	Exponente	r = 2 ** 6 # r es 64
/	División	r = 3.5 / 2 # r es 1.75
//	División entera	r = 3.5 // 2 # r es 1.0
%	Módulo	r = 7 % 2 # r es 1

Tab. 2.1 *Operadores Aritméticos.*

No obstante hay que tener en cuenta que si utilizamos dos operandos enteros, Python determinará que queremos que la variable resultado también sea un entero, por lo que el resultado de, por ejemplo, 3 / 2 y 3 // 2 sería el mismo: 1. Si quisiéramos obtener los decimales necesitaríamos que al menos uno de los operandos fuera un número real, bien indicando los decimales o bien utilizando la función float. Esto es así porque cuando se mezclan tipos de números, Python convierte todos los operandos al tipo más complejo de entre los tipos de los operandos.

```
r = 3.0 / 2
r = float(3) / 2
```

Operadores a Nivel de Bit. Estos son operadores que actúan sobre las representaciones en binario de los operandos. Por ejemplo, si se ve una operación como 3 & 2, lo que se esta viendo es un and bit a bit entre los números binarios 11 y 10 (las representaciones en binario de 3 y 2).

El operador *and* (&), del inglés “y”, devuelve 1 si el primer bit operando es 1 **y** el segundo bit operando es 1. Se devuelve 0 en caso contrario. El resultado de aplicar and bit a bit a 11 y 10 sería entonces el número binario 10, o lo que es lo mismo, 2 en decimal (el primer dígito es 1 para ambas cifras, mientras que el segundo es 1 sólo para una de ellas).

El operador *or* (|), del inglés “o”, devuelve 1 si el primer operando es 1 **o** el segundo operando es 1. Para el resto de casos se devuelve 0. El operador *xor* u or exclusivo (^) devuelve 1 si uno de los operandos es 1 y el otro no lo es. El operador *not* (~), del inglés “no”, sirve para negar uno a uno cada bit; es decir, si el operando es 0, cambia a 1 y si es 1, cambia a 0. Por último los operadores de desplazamiento (<< y >>) sirven para desplazar los bits n posiciones hacia la izquierda o la derecha.

Operador	Descripción	Ejemplo
&	and	r = 3 & 2 # r es 2
|	or	r = 3 | 2 # r es 3
^	xor	r = 3 ^ 2 # r es 1
~	not	r = ~3 # r es -4
<<	Desplazamiento izq.	r = 3 << 1 # r es 6
>>	Desplazamiento der.	r = 3 >> 1 # r es 1

Tab. 2.2 *Operadores a nivel de bit.*

Sentencias condicionales. Los condicionales comprueban condiciones y hacer que nuestro programa se comporte de una forma u otra, que ejecute un fragmento de código u otro, dependiendo de esta condición. Aquí es donde cobran su

importancia el tipo booleano y los operadores lógicos y relacionales sobre los tipos básicos de Python.

If. La forma más simple de un estamento condicional es un if (del inglés si) seguido de la condición a evaluar, dos puntos (:) y en la siguiente línea e indentado, el código a ejecutar en caso de que se cumpla dicha condición.

```
fav = "mundogeek.net"
# si (if) fav es igual a "mundogeek.net"
if fav == "mundogeek.net":
print "Tienes buen gusto!"
print "Gracias"
```

if … else. La condición if else algo más complicado. Sin duda podríamos añadir otro if que tuviera como condición la negación del primero.

```
if fav == "mundogeek.net":
print "Tienes buen gusto!"
print "Gracias"
if fav != "mundogeek.net":
print "Vaya, que lástima"
```

if … elif … elif … else. La condición queda una construcción más que ver, que es la que hace uso del elif.

```
 if numero < 0:
 print "Negativo"
 elif numero > 0:
 print "Positivo"
 else:
 print "Cero"
```

elif es una contracción de *else if*, por lo tanto elif numero > 0 puede leerse como "si no, si numero es mayor que 0". Es decir, primero se evalúa la condición del if. Si es cierta, se ejecuta su código y se continúa ejecutando el código posterior al condicional; si no se cumple, se evalúa la condición del elif.

Si se cumple la condición del elif se ejecuta su código y se continúa ejecutando el código posterior al condicional; si no se cumple y hay más de un elif se continúa con el siguiente en orden de aparición. Si no se cumple la condición del if ni de ninguno de los elif, se ejecuta el código del else.

A if C else B. También existe una construcción similar al operador de otros lenguajes, que no es más que una forma compacta de expresar un if else. En esta construcción se evalúa el predicado C y se devuelve A si se cumple o B si no se cumple: *A* if *C* else *B*. Veamos un ejemplo:

```
var = "par" if (num % 2 == 0) else "impar"
```

En Python no existe la construcción del switch, que podría emularse con un simple diccionario, así que pasemos directamente a los bucles.

Cadenas. Las cadenas son un texto encerrado entre comillas simples (‘cadena’) o dobles (“cadena”). Dentro de las comillas se pueden añadir caracteres especiales escapándolos con \, como \n, el carácter de nueva línea, o \t, el de tabulación. Una cadena puede estar precedida por el carácter u o el carácter r, los cuales indican, respectivamente, que se trata de una cadena que utiliza codificación Unicode y una cadena *raw* (del inglés, cruda).

Las cadenas raw se distinguen de las normales en que los caracteres escapados mediante la barra invertida (\) no se sustituyen por sus contrapartidas. Esto es especialmente útil, por ejemplo, para las expresiones regulares, como veremos en el capítulo correspondiente.

```
unicode = u”äóè”
raw = r”\n”
```

También es posible encerrar una cadena entre triples comillas (simples o dobles). De esta forma podremos escribir el texto en varias líneas, y al imprimir la cadena, se respetarán los saltos de línea que introdujimos sin tener que recurrir al carácter \n, así como las comillas sin tener que escaparlas. Triple = “““primera línea, esto se vera en otra linea”””

Las cadenas también admiten operadores como +, que funciona realizando una concatenación de las cadenas utilizadas como operandos y *, en la que se repite la cadena tantas veces como lo indique el número utilizado como segundo operando.

```
a = “uno”
b = “dos”
c = a + b # c es “unodos”
c = a * 3 # c es “unounouno”
```

Tuplas. Todo lo que hemos explicado sobre las listas se aplica también a las tuplas, a excepción de la forma de definirla, para lo que se utilizan paréntesis en lugar de corchetes.

```
t = (1, 2, True, “python”)
```

En realidad el constructor de la tupla es la coma, no el paréntesis, pero el intérprete muestra los paréntesis, y nosotros deberíamos utilizarlos, por claridad. Además hay que tener en cuenta que es necesario añadir una coma para tuplas de un solo elemento, para diferenciarlo de un elemento entre paréntesis.

```
>>> t = 1, 2, 3
>>> type(t)
type “tuple”
```

```
>>> t = (1)
>>> type(t)

type "int"
>>> t = (1,)
>>> type(t)
type "tuple"
```

Para referirnos a elementos de una tupla, como en una lista, se usa el operador []: Podemos utilizar el operador [] debido a que las tuplas, al igual que las listas, forman parte de un tipo de objetos llamados secuencias.

```
mi_var = t[0] # mi_var es 1
mi_var = t[0:2] # mi_var es (1, 2)
```

for … in. En Python for se utiliza como una forma genérica de iterar sobre una secuencia. Y como tal intenta facilitar su uso para este fin. Este es el aspecto de un bucle for en Python

```
secuencia = ["uno", "dos", "tres"]
for elemento in secuencia:
print elemento
```

Los for se utilizan en Python para recorrer secuencias, por lo que vamos a utilizar un tipo secuencia, como es la lista, para nuestro ejemplo. Leamos la cabecera del bucle como si de lenguaje natural se tratara: "para cada elemento en secuencia". Y esto es exactamente lo que hace el bucle: para cada elemento que tengamos en la secuencia, ejecuta estas líneas de código.

Lo que hace la cabecera del bucle es obtener el siguiente elemento de la secuencia secuencia y almacenarlo en una variable de nombre elemento. Por esta razón en la primera iteración del bucle elemento valdrá "uno", en la segunda "dos", y en la tercera "tres".

Funciones. Una función es un fragmento de código con un nombre asociado que realiza una serie de tareas y devuelve un valor. A los fragmentos de código que tienen un nombre asociado y no devuelven valores se les suele llamar procedimientos. En Python no existen los procedimientos, ya que cuando el programador no especifica un valor de retorno la función devuelve el valor None (nada), equivalente al null de Java.

Además de ayudarnos a programar y depurar dividiendo el programa en partes las funciones también permiten reutilizar código. En Python las funciones se declaran de la siguiente forma.

```
def mi_funcion(param1, param2):
print param1
print param2
```

Es decir, la palabra clave def seguida del nombre de la función y entre paréntesis los argumentos separados por comas. A continuación, en otra línea, indentado y después de los dos puntos tendríamos las líneas de código que conforman el código a ejecutar por la función. También podemos encontrarnos con una cadena de texto como primera línea del cuerpo de la función. Estas cadenas se conocen con el nombre de *docstring* (cadena de documentación) y sirven, como su nombre indica, a modo de documentación de la función.

```
def mi_funcion(param1, param2):
"""Esta funcion imprime los dos valores pasados
como parametros"""
print param1
print param2
```

El operador de iPython o la función help del lenguaje para proporcionar una ayuda sobre el uso y utilidad de las funciones. Todos los objetos pueden tener docstrings, no solo las funciones, como veremos más adelante. Volviendo a la declaración de funciones, es importante aclarar que al declarar la función lo único que hacemos es asociar un nombre al fragmento de código que conforma la función, de forma que podamos ejecutar dicho código más tarde referenciándolo por su nombre.

```
mi_funcion("hola", 2)
```

Es decir, el nombre de la función a la que queremos llamar seguido de los valores que queramos pasar como parámetros entre paréntesis. La asociación de los parámetros y los valores pasados a la función se hace normalmente de izquierda a derecha: como a param1 le hemos dado un valor "hola" y param2 vale 2, mi_funcion imprimiría hola en una línea, y a continuación 2.

Sin embargo también es posible modificar el orden de los parámetros si indicamos el nombre del parámetro al que asociar el valor a la hora de llamar a la función, El número de valores que se pasan como parámetro al llamar a la función tiene que coincidir con el número de parámetros que la función acepta según la declaración de la función. En caso contrario Python se quejará.

```
mi_funcion(param2 = 2, param1 = "hola")
>>> mi_funcion("hola")
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: mi_funcion() takes exactly 2 arguments (1
given)
```

También es posible, no obstante, definir funciones con un número variable de argumentos, o bien asignar valores por defecto a los parámetros para el caso de que no se indique ningún valor para ese parámetro al llamar a la función. Los valores por defecto para los parámetros se definen situando un signo igual después del nombre del parámetro y a continuación el valor por defecto.

```
def imprimir(texto, veces = 1):
print veces * texto
```

En el ejemplo anterior si no indicamos un valor para el segundo parámetro se imprimirá una sola vez la cadena que le pasamos como primer parámetro, si se le indica otro valor, será este el que se utilice, Para definir funciones con un número variable de argumentos colocamos un último parámetro para la función cuyo nombre debe precederse de un signo *.

```
>>> imprimir("hola")
hola
>>> imprimir("hola", 2)
holahola
def varios(param1, param2, *otros):
for val in otros:
print val
varios(1, 2)
varios(1, 2, 3)
varios(1, 2, 3, 4)
```

Esta sintaxis funciona creando una tupla (de nombre otros en el ejemplo) en la que se almacenan los valores de todos los parámetros extra pasados como argumento. Para la primera llamada, varios(1, 2), la tupla otros estaría vacía dado que no se han pasado más parámetros que los dos definidos por defecto, por lo tanto no se imprimiría nada. En la segunda llamada otros valdría (3,), y en la tercera (3, 4). También se puede preceder el nombre del último parámetro con **, en cuyo caso en lugar de una tupla se utilizaría un diccionario.

Orientación a objetos. La Programación Orientada a Objetos (POO u OOP según sus siglas en inglés) es un paradigma de programación en el que los conceptos del mundo real relevantes para nuestro problema se modelan a través de clases y objetos, y en el que nuestro programa consiste en una serie de interacciones entre estos objetos.

Clases y objetos. Para entender este paradigma primero tenemos que comprender qué es una clase y qué es un objeto. Un objeto es una entidad que agrupa un estado y una funcionalidad relacionados. El estado del objeto se define a través de variables llamadas atributos, mientras que la funcionalidad se modela a través de funciones a las que se les conoce con el nombre de métodos del objeto.

Un ejemplo de objeto podría ser un coche, en el que tendríamos atributos como la marca, el número de puertas o el tipo de carburante y métodos como arrancar y parar. O bien cualquier otra combinación de atributos y métodos según lo que fuera relevante para nuestro programa.

Una clase es una plantilla genérica a partir de la cuál instanciar los objetos; plantilla que es la que define qué atributos y métodos tendrán los objetos de esa clase. Volviendo a nuestro ejemplo: en el mundo real existe un conjunto de objetos

a los que llamamos coches y que tienen un conjunto de atributos comunes y un comportamiento común, esto es a lo que llamamos clase. Sin embargo, mi coche no es igual que el coche de mi vecino, y aunque pertenecen a la misma clase de objetos, son objetos distintos.

En Python las clases se definen mediante la palabra clave class seguida del nombre de la clase, dos puntos (:) y a continuación, indentado, el cuerpo de la clase. Como en el caso de las funciones, si la primera línea del cuerpo se trata de una cadena de texto, esta será la cadena de documentación de la clase o docstring.

```
class Coche:
"""Abstraccion de los objetos coche."""
def __init__(self, gasolina):
self.gasolina = gasolina
print "Tenemos", gasolina, "litros"
def arrancar(self):
if self.gasolina > 0:
print "Arranca"
else:
print "No arranca"
def conducir(self):
if self.gasolina > 0:
self.gasolina -= 1
print "Quedan", self.gasolina, "litros"
else:
print "No se mueve"
```

El método __init__, con una doble barra baja al principio y final del nombre, se ejecuta justo después de crear un nuevo objeto a partir de la clase, proceso que se conoce con el nombre de instanciación. El método __init__ sirve, como sugiere su nombre, para realizar cualquier proceso de inicialización que sea necesario.

Iteraciones de orden superior sobre listas. Una de las cosas más interesantes que podemos hacer con nuestras funciones de orden superior es pasarlas como argumentos de las funciones map, filter y reduce. Estas funciones nos permiten sustituir los bucles típicos de los lenguajes imperativos mediante construcciones equivalentes.

Map(function, sequence[, sequence, ...]). La función map aplica una función a cada elemento de una secuencia y devuelve una lista con el resultado de aplicar la función a cada elemento. Si se pasan como parámetros n secuencias, la función tendrá que aceptar n argumentos. Si alguna de las secuencias es más pequeña que las demás, el valor que le llega a la función function para posiciones mayores que el tamaño de dicha secuencia será None. A continuación podemos ver un ejemplo en el que se utiliza map para elevar al cuadrado todos los elementos de una lista.

```
def cuadrado(n):
return n ** 2
l = [1, 2, 3]
l2 = map(cuadrado, l)
```

Filter(function, sequence). La función filter verifica que los elementos de una secuencia cumplan una determinada condición, devolviendo una secuencia con los elementos que cumplen esa condición. Es decir, para cada elemento de sequence se aplica la función function; si el resultado es True se añade a la lista y en caso contrario se descarta. A continuación podemos ver un ejemplo en el que se utiliza filter para conservar solo los números que son pares.

```
def es_par(n):
return (n % 2.0 == 0)
l = [1, 2, 3]
```

Funciones lambda. El operador lambda sirve para crear funciones anónimas en línea. Al ser funciones anónimas, es decir, sin nombre, estas no podrán ser referenciadas más tarde. Las funciones lambda se construyen mediante el operador lambda, los parámetros de la función separados por comas (atención, SIN paréntesis), dos puntos (:) y el código de la función.

Esta construcción podría haber sido de utilidad en los ejemplos anteriores para reducir código. El programa que utilizamos para explicar filter, por ejemplo, podría expresarse así.

```
l = [1, 2, 3]
l2 = filter(lambda n: n % 2.0 == 0, l)
def es_par(n):
return (n % 2.0 == 0)
l = [1, 2, 3]
l2 = filter(es_par, l)
```

La forma más sencilla de serializar un objeto usando pickle es mediante una llamada a la función dump pasando como argumento el objeto a serializar y un objeto archivo en el que guardarlo (o cualquier otro tipo de objeto similar a un archivo, siempre que ofrezca métodos read, realine y write).

```
try:
import cPickle as pickle
except ImportError:
import pickle
fichero = file(“datos.dat”, “w”)
animales = [“piton”, “mono”, “camello”]
pickle.dump(animales, fichero)
fichero.close()
```

La función dump también tiene un parámetro opcional protocol que indica el protocolo a utilizar al guardar. Por defecto su valor es 0, que utiliza formato texto y

es el menos eficiente. El protocolo 1 es más eficiente que el 0, pero menos que el 2. Tanto el protocolo 1 como el 2 utilizan un formato binario para guardar los datos.

Crear ejecutables .exe. Tanto en Mac OS como en la mayor parte de las distribuciones Linux el intérprete de Python está instalado por defecto, por lo que los usuarios de estos sistemas no tienen mayor complicación a la hora de instalar y ejecutar aplicaciones escritas en Python.

En el caso de Windows, esto no es así, por lo que sería interesante que los usuarios de este sistema operativo no tuvieran que instalar el intérprete de Python. También sería interesante que nuestro programa consistiera en un archivo .exe en lugar de uno o varios archivos .py, para simplificar las cosas.

Todo esto se logra gracias a py2exe, una extensión para distutils que, como su nombre indica, permite crear ejecutables para Windows a partir de código Python, y que permite ejecutar estas aplicaciones sin necesidad de tener instalado el intérprete de Python en el sistema.

Py2exe funciona examinando nuestro código fuente en busca de los módulos y paquetes que utilizamos, compilándolos y construyendo un nuevo archivo que incluye estos archivos y un pequeño intérprete de Python integrado. Para probar el funcionamiento de py2exe creemos un pequeño programa ejemplo.py.

```
print "Soy un .exe"
```

El archivo setup.py correspondiente. Los cambios que tenemos que realizar a setup.py son sencillos: importar py2exe, y utilizar los argumentos console y windows para indicar el nombre del script o scripts que queramos convertir en ejecutables de consola o ejecutables de interfaz gráfica, respectivamente.

```
from distutils.core import setup
import py2exe
setup(name="Aplicacion de ejemplo",
versión="0.1",
description="Ejemplo del funcionamiento de distutils",
author="Raul Gonzalez",
author_email="zootropo en gmail",
url="http://mundogeek.net/tutorial-python/",
license="GPL",
scripts=["ejemplo.py"],
console=["ejemplo.py"])
```

Para crear el ejecutable, utilizamos una nueva opción de línea de comandos para setup.py disponible tras importar el módulo y llamada, cómo no, py2exe:

```
python setup.py py2exe
```

Build, con las librerías compiladas, y un directorio dist, con los archivos que conforman nuestra aplicación. Entre los archivos que podemos encontrar en dist tendremos uno o varios ejecutables con el mismo nombre que los scripts indicados en console y windows, un archivo python*.dll, que es el intérprete de Python, y un archivo library.zip, que contiene varios archivos pyc que son los módulos que utiliza la aplicación compilados.

Para reducir el número de archivos a distribuir, se utiliza la opción --bundle de py2exe para añadir a library.zip las dll y los pyd (--bundle 2) o las dll, los pyd y el intérprete (--bundle 1). Podemos añadir un nuevo argumento options a la función setup que indique el valor a utilizar (opción bundle_files), de forma que no tengamos que añadir el flag --bundle cada vez que usemos el comando py2exe.

```
python setup.py py2exe --bundle 1
from distutils.core import setup
import py2exe
setup(name=”Aplicacion de ejemplo”,
      version=”0.1”,
      description=”Ejemplo del funcionamiento de distutils”,
      author=”Raul Gonzalez”,
      author_email=”zootropo en gmail”,
      url=”http://mundogeek.net/tutorial-python/”,
      license=”GPL”,
      scripts=[“ejemplo.py”],
      console=[“ejemplo.py”],
      options={“py2exe”: {“bundle_files”: 1}})
```

Por último podemos incluso prescindir de library.zip e incrustarlo en el ejecutable utilizando el argumento zipfile=None.

```
from distutils.core import setup
import py2exe
setup(name=”Aplicacion de ejemplo”,
     version=”0.1”,
     description=”Ejemplo del funcionamiento de distutils”,
     author=”Raul Gonzalez”,
     author_email=”zootropo en gmail”,
     url=”http://mundogeek.net/tutorial-python/”,
     license=”GPL”,
     scripts=[“ejemplo.py”],
     console=[“ejemplo.py”],
     options={“py2exe”: {“bundle_files”: 1}},
     zipfile=None)
```

2.2 Glade

Glade es el diseñador de interfaces gráficas para GTK. Permite la creación de interfaces gráficas de usuario de manera visual, el Código generado puede ser en uno de varios lenguajes. Glade es un desarrollador de interfaces Permite construir

de forma gráfica e interactiva interfaces de usuario gráficos para Gnome/Gtk, publicado bajo la licencia GNU GPL. Glade también permite definir los nombres de los handler (funciones) que se asociarán a cada uno de los eventos del interfaz.

Una de las claras ventajas de Glade es la generación del fichero XML que permite cargar la interfaz gráfica en tiempo de ejecución desde distintos lenguajes de programación. Distintos proyectos de software libre utilizan Glade para generar las interfaces de usuario, un ejemplo es gtkPod el popular programa que emula al iTunes de Apple y que permite sincronizar el iPod con nuestro PC bajo GNU/Linux.

Si a todo esto sumamos que Glade es multiplataforma y que es software libre, estamos ante un complemento ideal para el desarrollo de software con interfaz de usuario, especialmente en GNU/Linux y para la integración con el entorno de escritorio GNOME.

Fig. 2.3 *Ventana principal Glade.*

La Librería libGlade. Una vez hemos creado visualmente el interfaz deberemos usarlo en nuestros programas, para esto glade nos brinda dos opciones: La primera forma es que el propio Glade genere el código en C que crea el interfaz, código en el que posteriormente uniremos con nuestro rograma, esta opción suele ser poco recomendable porque es un poco engorrosa.

La segunda manera de hacerlo es usar Glade para que genere un fichero en el cual se describe con XML el interfaz. Este fichero nos servirá para especificárselo a la librería libglade, la cual, mediante un par de llamadas ejecutara el código necesario para usar el interfaz en nuestro programa.

GTK(GIMP Toolkit) es una biblioteca para crear interfaces graficas de usuario. Su licencia es la LGPL, así que mediante GTK podría desarrollar programas con licencias abiertas, gratuitas, libres y hasta licencias comerciales no libres sin mayores problemas. GTK está construido encima de GDK (GIMP Drawing Kit) que básicamente es un recubrimiento de las funciones de bajo nivel que deben haber para acceder al sistema de ventanas sobre el que se programe.

Se llama el UIMP toolkit porque fue escrito para el desarrollo del General Image Manipulation Program (GIMP), pero ahora GTK se utiliza en un gran número de

proyectos de programación, incluyendo el proyecto GNU Network. Object Model Environment (GNOME).

GTK es esencialmente una interfaz para la programación de aplicaciones orientadas a objetos (API). Aunque esta completamente escrito en C, esta implementado haciendo uso de la idea de clases y de funciones respuesta o de callback(punteros o funciones).

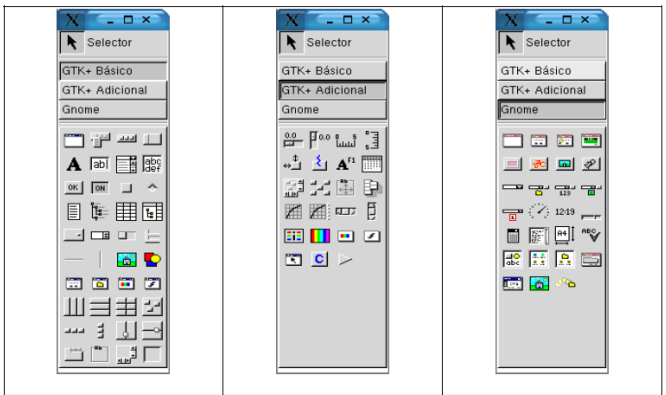
TCL (Tool Command Languaje) es un lenguaje de programación interpretado y multiplataforma. Es distribuido de forma totalmente gratuita, aunque su uso sea para aplicaciones comerciales, a través de Internet. Una de sus principales características es su gran facilidad con la que se pueden implementar funciones en C/C++ que pasan a ser nuevas instrucciones del intérprete. La extensión más conocida, y que es distribuida junto con el propio TCL, es TK(Tool Kit).

QTK. Es una herramienta construida a partir de TCL/TK, que permite a los diseñadores de interfaces de usuarios adoptar un enfoque basado en modelos rentables para el diseño de interfaces de usuario ejecutables. En esta herramienta los widgets pueden ser manejados y controlados dinámicamente y así facilita el desarrollo de las aplicaciones de las interfaces de usuario. El modulo QTK esta basado en el uso de descripciones de usuario.

La Ventana principal muestra los widgets que vayamos añadiendo a nuestra aplicación. (en la imagen superior no se ha añadido ninguno todavía). Por *widget* entenderemos cualquier componente que queramos incluir en la interfaz, por ejemplo Botones, Barra de menú, Barras de progreso, Cuadro de diálogo, etc. A lo largo de este manual mantendremos dicho término y no será traducido al castellano.

La Paleta de Widgets permite seleccionar qué elementos agregaremos a nuestra aplicación. En el ejemplo que presentaremos más adelante comenzaremos con una ventana. Para ello pulsamos en primer lugar sobre el icono de la esquina superior izquierda de la Paleta de Widgets Básicos. y sobre ella iremos añadiendo el resto de elementos, como indicaremos en el ejemplo.

El Editor de propiedades se activa cuando un widget esta seleccionado y permite modificar sus atributos (en la imagen superior permanece inactivo porque aún no hemos incluido ningún widget en nuestra interfaz).



Widgets GTK+ Básicos	Widgets GTK+ Adicionales	Widgets GTK+ Gnome
Widget más habituales en el desarrollo de la interfaz. Son los únicos que utilizaremos en este manual.	Aquí se presentan widgets algo más especiíficos y no serán discutidos en este manual.	Presenta los widgets propios del desarrollo de aplicaciones Gnome. Tampoco los discutiremos en este manual.

Fig. 2.4 *Paletas Widgets.*

Podemos definir un widget como un elemento de la interfaz gráfica de usuario. Un botón, una caja de texto, un desplegable y una etiqueta son ejemplos de widgets. GTK+ nos ofrece un rico conjunto de widgets, además de los mencionados contamos, entre otros, con barras de scroll horizontal y vertical, pestañas de separación, diferentes tipos de botones, botones de selección múltiple y botones de única selección.

Eventos. Cada uno de estos Widgets está asociado al menos a un evento, así que debemos dedicar unas líneas a este concepto. Siempre que pulsamos una tecla o presionamos algún botón del ratón se produce un evento y se emite una señal. Los Widgets de la interfaz deben respoder a uno o más eventos.

Por ejemplo, podemos diseñar un elemento de un menú que se llame *Salir*, y precisamente queramos que la aplicación responda de una determinada manera al pulsar sobre dicho elemento (terminando la ejecución de la aplicación en este caso).

Hay muchísimos eventos y señales, y la mayoría de ellos van a ser ignorados por la aplicación (por ejemplo, el teclado no va a jugar ningún papel en la aplicación que vamos a desarrollar aquí, así que las pulsaciones de las teclas serán simplemente ignoradas). Si queremos que un elemento de la interfaz responda a un determinado evento, debemos escribir una función que indique al programa la respuesta que queremos que ofrezca a tal acción.

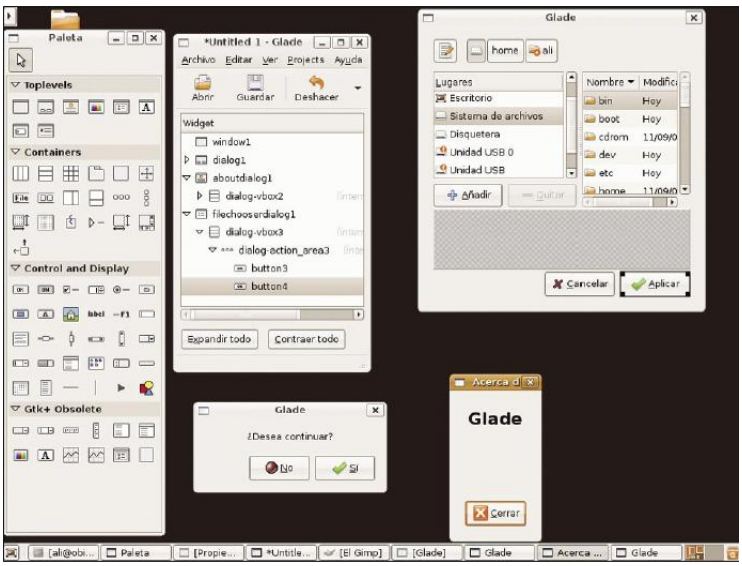


Fig. 2.5 *cuadros de dialogo que se pueden crear en Glade.*

Tipos de datos. Los tipos de datos (char, int, float, double), pueden diferir entre los diferentes SO ó arquitecturas de hardware. Glib libera al programador de

prestar atención a estos detalles y en su lugar ofrece un conjunto de tipos propios (gchar, gint, gint32, guint, guint32, guchar, etc).

Gestión de memoria. La gestión de memoria es una tarea especialmente delicada, por lo que Glib la administra automáticamente. Si el programador necesita mayor control de la memoria entonces tendrá a su disposición gran cantidad de métodos que le permitirán un control más exhaustivo de ésta.

Estructuras de datos. Las estructuras de datos más comunes como listas enlazadas, arreglos dinámicos, tablas de tablas de claves, pilas y colas, ya están disponibles en Glib. Sistema de objetos: El sistema de Objetos de Glib se ha diseñado con miras a flexibilidad y capacidad de adaptarse a las necesidades del usuario. Se ha creado con la intención de integrarse fácilmente con otros lenguajes de programación diferentes de C. Todos los objetos de GTK+ se derivan de la clase fundamental de Glib: Gobject.

Bucle de ejecución. En lugar de crear un bucle de eventos nosotros mismos, podemos dejarle la tarea a Glib para centrarnos en el diseño y desarrollo de nuestras aplicaciones asíncronas. Glib se encarga de la distribución de señales y mensajes a las diferentes partes de nuestro programa. También se encarga de administrar alarmas (temporizadores para aplicaciones síncronas), momentos de inactividad de la aplicación, eventos de entrada y salida en tuberías, sockets, o descriptores de archivos, así como hilos.

Macros de operaciones matemáticas sencillas. Existen ciertas operaciones matemáticas comunes que no se encuentran disponibles en la biblioteca estándar de C. MIN(a,b) y MAX(a,b) calculan el valor mínimo y máximo de entre dos números a y b, mientras que ABS(n) calcula el valor absoluto de un número n.

CLAMP(x,a,b) se asegura de que el número x se encuentre dentro de los límites a y b. Si x se encuentra dentro de estos límites, CLAMP() regresará el número x, si esto no se cumple y x es mayor que el límite superior b, CLAMP() regresará este valor, de lo contrario (x es menor que el límite inferior a), CLAMP() regresará el valor de límite inferior a. Esta macro resulta confusa, pero es útil al posicionar objetos gráficos en la pantalla y simular cierta resistencia al movimiento

Macros para verificación de errores excepciones y depurado. Un buen diseño de software no viene de la noche a la mañana. Parte importante del tiempo de desarrollo de un programa se consume en la depuración de errores. También es cierto que parte importante del total del código fuente escrito de un programa robusto se dedica a la validación y corrección de posibles errores, es decir, que las cosas que deban estar en orden realmente lo estén.

Los desarrolladores de Glib nos ofrecen diferentes herramientas 7 macros para ayudarnos a mejorar nuestros programas. g_assert() recibe como parámetro una expresión, tal y como se usa en el condicional if... then ... else ... Si la condición especificada falla o es FALSE, el programa termina especificando un mensaje de error. Por ejemplo la siguiente instrucción.

```
#include <glib.h>
/* ... */
g_assert(puerto == ABIERTO);
/* ... */ ...terminará el programa con un mensaje de
error “Assertion puerto==ABIERTO
failed”, si la variable puerto es falsa.
```

El complemento de g_assert() es g_assert_not_reached(). Esta macro termina el programa y despliega un mensaje de error si alguna vez se llega a ella. Un buen ejemplo de aplicación de estas macros se daría en un hipotética función que transforma cadenas provenientes, por ejemplo, de una comunicación serial.

```
void transformo_cadenas (gchar *cadena) {
/* ... */
g_assert (cadena != NULL);
/* ... */
}
g_assert()
```

Comprobará si el contenido de la variable cadena está vacío, de ser así interrumpirá el programa impidiendo que más errores se propaguen. Estas macros pueden desactivarse en compilaciones finales mediante la definición de g_disable_assert al momento de compilar la aplicación. g_return_if_fail() toma una expresión y regresa de la función si tal expresión no resulta verdadera o true. De lo contrario registra un mensaje de aviso y regresa de la función. g_return_if_fail() sólo se puede utilizar en funciones que no regresan ningún valor.

Para aquellas funciones que debe regresar un valor, está g_return_val_if_fail(expr, val), que regresa el valor val en función del la expresión expr al igual que g_return_if_fail(). Las aplicaciones escritas en GTK+ usualmente necesitan pasar datos entre las diferentes partes del programa.

Glib define seis macros básicas de conversión de tipos, sin embargo, conforme avancemos veremos que habrá macros de conversión de tipo para casi cualquier objeto o widget que usemos. Como veremos más tarde, será común convertir un tipo de dato en otro. Generalmente referido como casting o moldeado en C, esta técnica permite que GTK+ se comporte como una librería orientada a Objetos.

La manera de pasar datos de una parte de la aplicación a otra generalmente se hace utilizando gpointer,el cual es lo suficientemente general como para pasar cualquier estructura de datos, sin embargo existe una limitante al querer pasar números en lugar de estructuras de datos. Si, por ejemplo, deseáramos pasar un número entero en lugar de una estructura de datos, deberíamos de hacer algo como esto.

```
gint *ip = g_new (int, 1);
*ip = 42;
```

Ahora tenemos un puntero a una constante de tipo gint. La desventaja de esto es que ahora nosotros tendremos que hacernos cargo de liberar la memoria del número entero. Los punteros siempre tienen un tamaño de al menos 32 bits (en las plataformas que Glib está portada). En base a esto podríamos tratar de asignar el valor que queremos pasar a un puntero.

```
gpointer p;
int i;
p = (void*) (long) 42;
i = (int) (long) p;
```


```
gpointer p;
int i;
p = (void*) (long) 42;
i = (int) (long) p;
```


Esto se vuelve demasiado complicado como para llevarlo a la práctica, por eso los desarrolladores de glib han creado las macros gint_to_pointer(), guint_to_pointer() y gsize_to_pointer() para empacar un gint, guint o gsize en un puntero de 32 bits. Análogamente gpointer_to_gint(), g_pointer_to_guint() y gpointer_to_gsize() sirven para obtener el número que se ha empacado en el puntero de 32 bits.

Bucle de ejecución y eventos. El bucle de eventos de GTK+ es el responsable de que el sistema de señales funcione correctamente, ya que el primero no es más que un bucle interno de GTK+, en el que se van, una y otra vez, comprobando los estados de cada uno de los elementos de la aplicación, e informando de dichos cambios a los elementos que se hayan registrado para ser informados.

Este bucle de eventos GTK+ se traduce básicamente en dos funciones, que son gtk_main() y gtk_main_quit(). gtk_main() entrega el control de cualquier programa al bucle de eventos de GTK+. Esto significa que, una vez que se haya realizado la llamada a gtk_main(), se cede todo el control de la aplicación a GTK+. Aunque gtk_main() toma el control de la aplicación, es posible ejecutar otras porciones de código aprovechando el sistema se señales usando algún manejador (instalado ANTES de llamar a gtk_main()).

Dentro de algún manejador o retrollamada se puede llamar a gtk_main_quit() que termina el bucle de eventos de GTK+. El pseudo-código de una típica aplicación GTK+ seria.

```
int main (int argc, char *argv[])
{
gtk_init (&argc, &argv);
/* creación del interfaz principal */
/* conexión a las distintas señales */
gtk_main ();
return 0;
}
```

Como puede comprobarse, el programa inicializa GTK+, crea el interfaz básico, conecta funciones a las distintas señales en las que esté interesado (llamadas a g_signal_connect()), para seguidamente entregar el control del programa a GTK+ mediante gtk_main(). Cuando en algún manejador de señal realicemos una llamada a gtk_main_quit(), gtk_main() retornará, tras lo cual la aplicación termina. los estados de cada uno de los elementos de la aplicación, e informando de dichos cambios a los elementos que se hayan registrado para ser informados.

Este bucle de eventos GTK+ se traduce básicamente en dos funciones, que son gtk_main() y gtk_main_quit(). gtk_main() entrega el control de cualquier programa al bucle de eventos de GTK+. Esto significa que, una vez que se haya realizado la llamada a gtk_main(), se cede todo el control de la aplicación a GTK+. Aunque gtk_main() toma el control de la aplicación, es posible ejecutar otras porciones de código aprovechando el sistema se señales usando algún manejador (instalado ANTES de llamar a gtk_main())

Dentro de algún manejador o retrollamada se puede llamar a gtk_main_quit() que termina el bucle de eventos de GTK+. El pseudo-código de una típica aplicación GTK+ seria.

```
int main (int argc, char *argv[])
{
gtk_init (&argc, &argv);
/* creación del interfaz principal */
/* conexión a las distintas señales */
gtk_main ();
return 0;
}
```

El programa inicializa GTK+, crea el interfaz básico, conecta funciones a las distintas señales en las que esté interesado (llamadas a g_signal_connect()), para seguidamente entregar el control del programa a GTK+ mediante gtk_main(). Cuando en algún manejador de señal realicemos una llamada a gtk_main_quit(), gtk_main() retornará, tras lo cual la aplicación termina

2.3 Arduino

Al ser Arduino una plataforma de hardware libre tanto su diseño como su distribución puede utilizarse libremente para el desarrollo de cualquier tipo de proyecto sin haber adquirido ninguna licencia. Por eso existen varios tipos de placa oficiales, las creadas por la comunidad Arduino o las no oficiales creadas por terceros pero con características similares. En la placa Arduino es donde conectaremos los sensores, actuadores y otros elementos necesarios para comunicarnos con el sistema.

Arduino es una herramienta para hacer que los ordenadores puedan sentir y controlar el mundo físico a través de tu ordenador personal. Es una plataforma de desarrollo de computación física (physical computing) de código abierto, basada

en una placa con un sencillo micro controlador y un entorno de desarrollo para crear software (programas) para la placa.

Está formado por una serie de menús, una barra de herramientas con botones para las funciones comunes, un editor de texto donde escribiremos el código, un área de mensajes y una consola de texto.

Arduino Ethernet Shield. La Arduino Ethernet Shield conecta el Arduino a Internet. Se conecta este módulo en la placa Arduino, conectarlo a la red con un cable RJ45 y seguir algunas instrucciones sencillas para empezar a controlar su mundo a través de internet. Todos los elementos de la plataforma de Arduino - hardware, software y documentación - son de libre acceso y de fuente abierta.

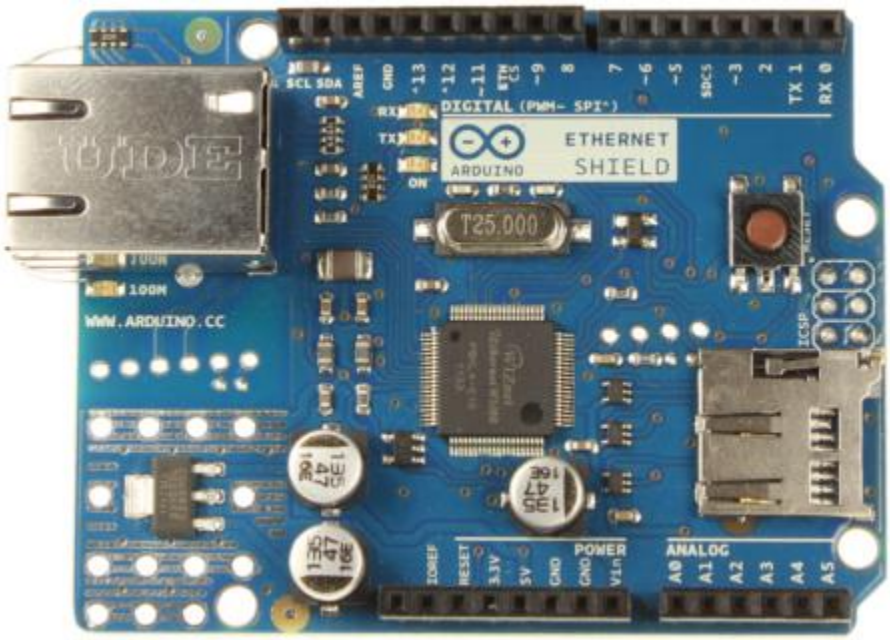


Fig. 2.6 *Arduino Ethernet Shield.*

Requiere una placa Arduino, 5V Tensión de funcionamiento (suministrado por la Junta Arduino), Ethernet Controller: W5100 con buffer interno de 16K, La velocidad de conexión: 10 / 100Mb, Conexión con Arduino en el puerto SPI.

El Arduino Ethernet Shield (Fig. 2.6) permite a una placa Arduino para conectarse a internet. Se basa en la Wiznet W5100 chip de ethernet (hoja de datos). El Wiznet W5100 proporciona una red (IP) apilar capaz de TCP y UDP. Soporta hasta cuatro conexiones de socket simultáneas. Utilice la biblioteca de Ethernet para escribir bocetos que se conectan a Internet a través de la pantalla. El escudo de Ethernet se conecta a una placa Arduino usando largas encabezados por arrollamiento de hilo que se extienden a través del escudo.

Esto mantiene la disposición de pines intacta y permite que otro escudo para ser apilados en la parte superior. La revisión más reciente de la Junta expone el pinout 1.0 en rev 3 de la placa Arduino UNO. El escudo tiene una conexión Ethernet RJ-45 estándar, con un transformador de línea integrada y alimentación a través de Ethernet habilitado.

Hay una ranura de tarjeta micro-SD a bordo, que se puede utilizar para almacenar archivos para servir a través de la red.Es compatible con el Arduino Uno y Mega (utilizando la librería Ethernet). El lector de tarjetas microSD a bordo es accesible a través de la Biblioteca SD. Cuando se trabaja con esta biblioteca, SS es el Pin 4. La revisión original del escudo contiene una ranura para tarjetas SD de tamaño completo; esto no es compatible.

El escudo también incluye un controlador de reajuste, para asegurar que el módulo Ethernet W5100 se restablece correctamente en el encendido. Las revisiones anteriores del escudo no eran compatibles con la Mega y necesitan reiniciar manualmente después del encendido. El escudo actual tiene una alimentación a través de Ethernet (PoE módulo) diseñado para extraer energía de un cable Ethernet de par trenzado de categoría 5 convencional.

IEEE802 .3af compatible, Ondulación baja producción y el ruido (100mVpp), Entrada rango de voltaje de 36V a 57V, Sobrecarga y cortocircuito protección, 9V de salida, Alta eficiencia convertidor DC / DC: tip 75% a 50% de carga, Aislamiento 1500 V (entrada a la salida), PWR: indica que la placa y el escudo son powered.

LINK. Indica la presencia de un enlace de red y parpadea cuando el escudo transmite o recibe datos, FULLD: indica que la conexión de red es full dúplex, 100M: indica la presencia de un Mb / s 100 conexión de red (en contraposición a 10 Mb / s), RX: Parpadea cuando el escudo recibe datos, TX: parpadea cuando el escudo envía datos, COLL: parpadea cuando se detectan colisiones de red.

El puente de soldadura marcada "INT" puede conectarse a permitir que la placa Arduino para recibir una notificación por interrupciones de eventos desde el W5100, pero esto no es apoyado por la librería Ethernet. El puente conecta el pin INT del W5100 para pin digital 2 de la Arduino.

Arduino Ethernet. La Arduino Ethernet es una placa electrónica basada en el ATmega328. Cuenta con 14 pines digitales de entrada / salida, 6 entradas analógicas, un 16 MHz oscilador de cristal, un RJ45 de conexión, un conector de alimentación, una cabecera ICSP, y un botón de reinicio.Los pines 10, 11, 12 y 13 están reservados para la conexión con el módulo Ethernet y no debe ser utilizado de otra manera.

Esto reduce el número de pines disponibles a 9, con 4 disponibles como salidas PWM.Una alimentación opcional sobre el módulo Ethernet se puede agregar a la tarjeta también.La Ethernet se diferencia de otras placas en que no tiene un chip

integrado controlador de USB a serie, pero tiene una interfaz Wiznet Ethernet. Esta es la misma interfaz que se encuentra en el escudo Ethernet.

Un lector de tarjetas microSD a bordo, que se puede utilizar para almacenar archivos para servir a través de la red, es accesible a través de la Biblioteca SD. Pin 10 está reservado para la interfaz Wiznet, SS para la tarjeta SD está en el pin 4.

La cabecera de programación en serie de 6 pines es compatible con el de serie USB adaptador y también con los cables USB FTDI o con Sparkfun y tableros Adafruit estilo FTDI básicos de USB a serial de descanso. Cuenta con soporte para rearme automático, permitiendo bocetos que se cargan sin necesidad de pulsar el botón de reinicio en el tablero.Cuando se conecta a un adaptador USB a Serial, el Arduino Ethernet es alimentado desde el adaptador.

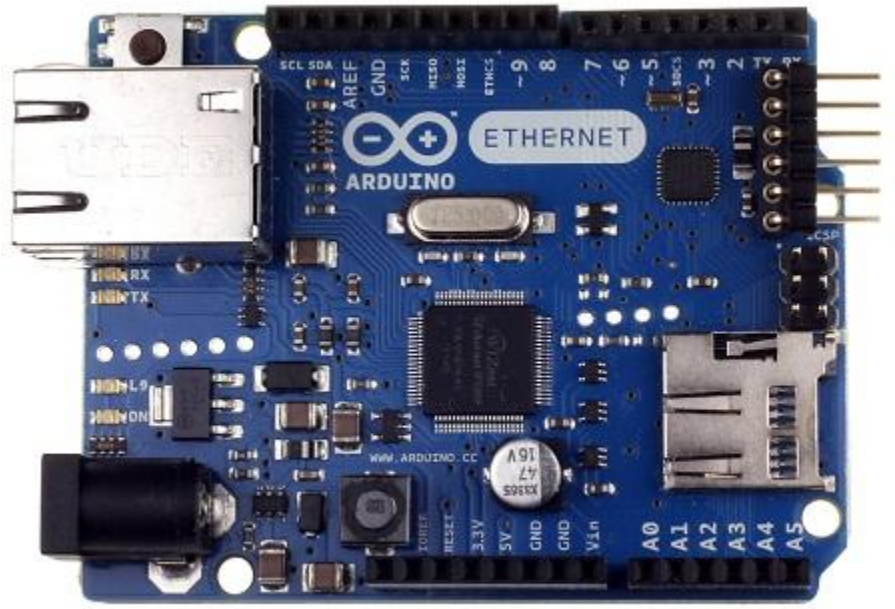


Fig. 2.7 *Arduino Ethernet revisión 3.*

La revisión 3 de la junta directiva introduce los 1,0 pinout estandarizados, que consiste en. SDA añadido y pines SCL que están cerca de la clavija de AREF y otros dos nuevos pasadores colocados cerca del pin RESET, esta será la oportunidad de escudo que I2C uso o componentes TWI para ser compatible con todas las placas Arduino;

La instrucción IOREF que permiten los escudos para adaptarse a la tensión suministrada desde la pizarra. Los escudos que utilizan el pin instrucción IOREF serán compatibles tanto con la placa que utilizan el AVR, que operan con 5V y con

el Arduino Debido que operan con 3.3V. Junto al pin instrucción IOREF hay un pin no está conectado, que se reserva para usos futuros.

Microcontroladores	ATmega328
Tensión de funcionamiento	5V
Plug Voltaje de entrada (recomendado)	7-12V
Tensión enchufe de entrada (límites)	6-20V
Voltaje de entrada PoE (límites)	36-57V
Digital pines I / O	14 (de las cuales 4 proporcionan salida PWM)
Arduino prendedores reservados:	
	10 a 13, utilizado para SPI
	4 utilizado para la tarjeta SD
F	2 W5100 de interrupción (cuando puente)
Pines de entrada analógica	6
Corriente DC por Pin I / O	40 mA
Corriente DC de 3.3V Pin	50 mA
Memoria Flash	32 KB (ATmega328) de los cuales 0,5 KB utilizado por el gestor de arranque
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Velocidad del reloj	16 MHz
Ethernet Controller Embedded W5100 TCP / IP	
Power Over Ethernet listomagnética Jack	
Tarjeta Micro SD, con traductores de voltaje activos	

Tab. 2.8 *Datos del arduino*

Energía. La placa también puede ser alimentado a través de una fuente de alimentación externa, una potencia opcional a través de Ethernet (PoE módulo), o mediante el uso de un conector Serial / cable USB FTDI.

La alimentación externa puede venir con un adaptador de CA a CC (pared-verruga) o la batería. El adaptador se puede conectar al conectar un enchufe de 2.1mm centro-positivo en el conector de alimentación de la placa. Los cables desde una batería se pueden insertar en los cabezales de pin GND y Vin del conector de alimentación.

El tablero puede funcionar con un suministro externo de 6 a 20 voltios. Si se suministra con menos de 7V, sin embargo, el pin de 5V puede suministrar menos de cinco voltios y la junta puede ser inestable. Si se utiliza más de 12 V, el

regulador de voltaje se puede sobrecalentar y dañar la placa. El rango recomendado es de 7 a 12 voltios.

Los pines de alimentación son como sigue: **VIN.** El voltaje de entrada a la placa Arduino cuando se trata de utilizar una fuente de alimentación externa (en oposición a 5 voltios de la conexión USB u otra fuente de alimentación regulada). Usted puede suministrar tensión a través de este pin, o, si el suministro de tensión a través de la toma de alimentación, acceso a él a través de este pin.

5V. Este pin como salida una 5V regulada del regulador en el tablero. El tablero puede ser alimentado ya sea desde la toma de alimentación de CC (7 - 12 V), el conector USB (5V), o por el pin VIN del tablero (7-12V). El suministro de tensión a través de los pines de 5V o 3.3V no pasa por el regulador, y puede dañar su tablero. No aconsejamos ella.

3V3. Un suministro de 3,3 voltios generada por el regulador de a bordo. Sorteo de corriente máxima es de 50 mA.

Instrucción IOREF. Este pin de la placa Arduino proporciona la referencia de tensión con la que opera el microcontrolador. Un escudo configurado puede leer el voltaje pin instrucción IOREF y seleccione la fuente de alimentación adecuada o habilitar traductores de voltaje en las salidas para trabajar con el 5V o 3.3V. El opcional PoE módulo está diseñado para extraer energía de un cable Ethernet de par trenzado de categoría 5 convencional:

IEEE802 .3af compatible, Ondulación baja producción y el ruido (100mVpp), Entrada rango de voltaje de 36V a 57V, Sobrecarga y cortocircuito protección, 9V de salida, Alta eficiencia convertidor DC / DC: tip 75% @ 50% de carga, Aislamiento 1500 V (entrada a la salida)

Entrada y Salida

Cada uno de los 14 pines digitales en el tablero de Ethernet se puede utilizar como una entrada o salida, usando pinMode () , digitalWrite () , y digitalRead () funciones. Funcionan a 5 voltios. Cada pin puede proporcionar o recibir un máximo de 40 mA y tiene una resistencia de pull-up (desconectado por defecto) de 20 a 50 kOhm. Además, algunos pines tienen funciones especializadas:

Interrupciones externas:. 2 y 3 Estos pines pueden configurarse para activar una interrupción en un valor bajo, un flanco ascendente o descendente, o un cambio en el valor. Ver el attachInterrupt () función para más detalles.

PWM:. 3, 5, 6, 9, 10 y proporcionar una salida PWM de 8 bits con el analogWrite () función. Led 9**,** Hay un LED incorporado conectado al pin digital 9. Cuando el pasador es de alto valor, el LED está encendido, cuando el pasador es bajo, es apagado. En la mayoría de otras placas Arduino, este LED se encuentra en el pin

13. Es en el pin 9 en el tablero de Ethernet porque pin 13 se utiliza como parte de la conexión SPI.

La tarjeta Ethernet tiene 6 entradas analógicas, etiquetado A0 a A5, cada uno de los cuales proporcionan 10 bits de resolución (es decir, 1.024 valores diferentes). Por defecto se miden desde el suelo a 5 voltios, aunque es posible cambiar el extremo superior de su rango usando el pin AREF y la analogReference función (). Además, algunos pines tienen funciones especializadas:

TWI:. A4 (SDA) y A5 (SCL) Apoyar la comunicación TWI utilizando la librería Wire . AREF**.** Voltaje de referencia para las entradas analógicas. Se utiliza con analogReference (). Restablecer**.** Esta línea BAJO para reajustar el microcontrolador. Normalmente se utiliza para añadir un botón de reinicio para escudos que bloquean el uno en el tablero.

Comunicación. La Arduino Ethernet tiene una serie de instalaciones para comunicarse con un ordenador, otro Arduino u otros microcontroladores. Una biblioteca Software Serial permite la comunicación en serie en cualquiera de los pines digitales del Uno.

El ATmega328 también soporta la comunicación TWI y SPI. El software de Arduino incluye una biblioteca de alambre para simplificar el uso del bus TWI; consulte la documentación para obtener más información. Para la comunicación SPI, utilice la librería SPI .La junta también se puede conectar a una red cableada a través de Ethernet. Cuando se conecta a una red, tendrá que proporcionar una dirección IP y una dirección MAC. La Biblioteca Ethernet es totalmente compatible. El lector de tarjetas microSD a bordo es accesible a través de la Biblioteca SD . Cuando se trabaja con esta biblioteca, SS está en el pin 4.

Programación. Es posible programar la placa Arduino Ethernet de dos maneras: a través de la cabecera de programación en serie de 6 pines, o con un programador ISP externo. La cabecera de programación en serie de 6 pines es compatible con los cables USB FTDI y la Sparkfun y tableros Adafruit estilo FTDI básicos de USB a serial de descanso entre ellos el conector USB-serie Arduino.

Cuenta con soporte para rearme automático, permitiendo bocetos que se cargan sin necesidad de pulsar el botón de reinicio en el tablero. Cuando se conecta a un adaptador USB-estilo FTDI, el Arduino Ethernet se apaga el adaptador. También puede pasar por alto el gestor de arranque y programar el microcontrolador a través del ICSP (In-Circuit Serial Programming) cabecea utilizando Arduino ISP o similar; ver estas instrucciones para más detalles.Todos los ejemplos de bocetos Ethernet funcionan como lo hacen con el escudo Ethernet.

Características Físicas. La longitud máxima y la anchura de la placa Ethernet son 2,7 y 2,1 pulgadas, respectivamente, con el RJ45 jack conector y el poder que se extiende más allá de la dimensión anterior. Cuatro orificios de los tornillos que la Junta pudiera estar unido a una superficie o caso. Tenga en cuenta que la distancia entre los pines digitales 7 y 8 es de 160 milésimas de pulgada (0,16 "), no un múltiplo par de la separación de 100 milésimas de pulgada de los otros pasadores.

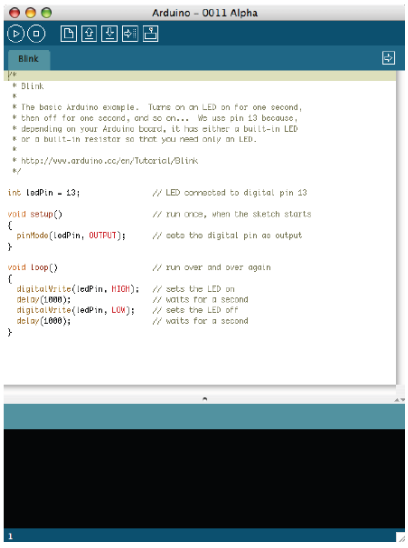


Fig. 2.7 *Entorno Arduino.*

Estructura de programación Zona global. Aquí será donde indicaremos a arduino los nombres de los pines y donde crearemos aquellas variables que queramos que existan en todo el programa. Aunque comprende todo lo que está fuera de las otras dos zonas, es recomendable agruparlo todo en la parte superior del código.

La función void setup (). Esta función se ejecuta cada vez que se inicia Arduino (incluyendo al pulsar RESET). Una de las operaciones que se realiza en **void setup ()** es la de configurar de los pines que vamos a utilizar.

La función void loop(). Esta función es el corazón de los programas creados con arduino. Es una función que permanece en ejecución en forma de bucle infinito. Esto quiere decir que se ejecuta de comienzo a fin, de forma repetida, siempre.

Fig. 2.7.1 *Estructura de programación.*

2.4 Interfaces

Es posible afirmar que los IED´s(Dispositivos electrónicos inteligentes) son el primer nivel en la integración de la automatización. Pero aún con las ventajas que dichos dispositivos proporcionan, hasta el momento solo encontramos “islas de automatización” esparcidas por la red eléctrica. Una cierta mejoría en la eficiencia se puede alcanzar al conectar los IED´s en un sencillo sistema de control integrado.

Más aún, la introducción de sistemas de control completamente integrados pueden llevarnos a una mayor eficiencia gracias a la redundancia de equipos, así como también menores costos de cableados, comunicaciones, operación y mantenimiento, así como una notable mejora en la calidad y confiabilidad en el suministro de la energía eléctrica.

A pesar de que los beneficios son bien conocidos, el enfoque de los sistemas de control integrados para la automatización de redes ha tenido pocos progresos en Venezuela, principalmente porque las interfaces de hardware y protocolos de los IED´s no están estandarizados. Los protocolos son tan numerosos como los proveedores, e incluso más, debido a que distintos productos del mismo proveedor a veces poseen diferentes protocolos.

Una solución a este problema es la instalación de compuertas que actúen como interfaces de hardware y de protocolos entre los IED´s y la red de área local (LAN, por sus siglas en inglés). Las compuertas permiten trabajar en una red común de comunicación y protocolo en toda la red con el objeto de integrar una gran cantidad de estos dispositivos.

Ellos suministran una interface física del sistema de control y automatización entre los IED´s (puertos RS232/ RS485) y la red eléctrica, a la vez que funcionan como convertidores de protocolos entre los dispositivos y la red estándar. Las compuertas hacen que todos los IED´s "luzcan" idénticos en cuanto a lo que la comunicación en la red se refiere.

Interfaz gráfica de usuario. Interfaz gráfica de usuario (En inglés Graphic User Interface, también conocido con su acrónimo GUI) es un método para facilitar la interacción del usuario con el ordenador o la computadora a través de la utilización de un conjunto de imágenes y objetos pictóricos (iconos, ventanas..) además de texto. Surge como evolución de la línea de comandos de los primeros sistemas operativos y es pieza fundamental en un entorno gráfico.

GUI es un acrónimo del vocablo inglés Graphical User Interface. Douglas Engelbart, además de inventor del ratón de ordenador, desarrolló la primera interfaz gráfica en los años 1960 en EE.UU. en los laboratorios de XEROX. Fue introducida posteriormente al público en las computadoras Apple Macintosh en 1984, y a las masas hasta 1993 con la primera versión popular del sistema operativo Windows 3.0.

La interface hombre-máquina puede llegar a ser el aspecto de mayor importancia dentro del sistema de control integrado de la red. Es a través de la interface que el operador controla y supervisa toda la red. Los datos deben ser presentados al operador de una manera clara y precisa. Debido a la naturaleza crítica de las acciones del operador respecto a los equipos de la red.

La tecnología a seleccionar aquí es la de la PC. La PC suministra una plataforma computacional bien poderosa para aplicaciones, a la vez que el software de interfaz gráfica con el usuario le permite servir como un medio avanzado de monitoreo y control para el operador de la red. Existen muchas tarjetas disponibles para la interfaz entre la PC con la LAN/WAN de la red. El rango de poder de la PC es variado a la vez que su costo es aceptable.

Corriendo en la computadora de la red se encuentra el software SCADA (Supervisory Control and Data Acquisition) recolectando datos desde los IED´s y almacenándolos en una base de datos central. Dichos datos podrán entonces ser accesados por el software de interfaz gráfica de usuario así como por cualquier software de aplicación.

La aplicación SCADA enviará cualquier comando de control ejecutado por el operador al IED seleccionado. La mayoría del software de interfaz gráfica de usuario disponible en el mercado puede ayudar al operador a supervisar y

controlar la red con gran eficiencia. La alta resolución y capacidad gráfica de muchos paquetes permite al operador la visualización de los datos de distintas formas (tabular, esquemática, p.e.).

2.5 Redes

Una red es básicamente una colección de computadores y otros dispositivos que pueden enviar y recibir datos entre ellos, más o menos en tiempo real. Una red se conecta normalmente por alambres, y los bits son convertidos a ondas electromagnéticas que viajan a través de los alambres. Sin embargo, también existen redes inalámbricas.

Estas transmiten datos a través de luz infrarroja o , microondas y cables de fibra óptica que envían luz visible a través de filamentos de vidrio. Dentro de la definición anterior se puede resaltar dos aspectos: el primero, la existencia dentro de la definición de la posibilidad de conexión de dispositivos a una red sin que estos tengan que ser necesariamente computadores.

El segundo, la posibilidad de transmitir información por medios no alambrados; estos dos puntos son fundamentales en el momento de describir las posibles aplicaciones, tanto actuales como futuras.

Topología de la Red. Dos enfoques pueden ser tomados en cuenta al usar las compuertas para la interface de la red. Por un lado, se usa una única compuerta multi-puertos como interface para múltiples IED´s, y por el otro, se usa una compuerta sencilla de bajo costo para cada dispositivo inteligente. El más económico dependerá de la ubicación de los dispositivos inteligentes.

Otro problema del cual se debe estar atento al integrar los IED´s en un sistema de control es la configuración de los dispositivos. Un gran número de dispositivos inteligentes sólo tiene un puerto de comunicaciones, el cual cumple con dos propósitos, escaneo de datos históricos en tiempo real, y acceso y/o almacenamiento de datos o archivos de datos.

La nota importante aquí es la siguiente: cuando se está reconfigurando un IED, los datos en tiempo real no están disponibles en el sistema. Esta pérdida puede ser crítica para el sistema si los datos se usan para aplicaciones en tiempo real. El sistema de control integrado debe ser capaz de reconocer y señalizar que la reconfiguración está en progreso y destacar que los datos en tiempo real están fuera de línea, habilitando tanto al operador como a cualquier aplicación a que se ajuste apropiadamente durante este proceso.

Muchos proveedores de IED´s han estado introduciendo al mercado productos con dos puertos, uno para el escaneo de datos históricos y en tiempo real, y el segundo para configuración. Estos dispositivos requieren de compuertas con dos puertos. Aunque son un medio elegante para lograr la interfaz entre la red de comunicación con los diferentes protocolos de los IED´s, las compuertas tienen sus desventajas.

Ellos incrementan el costo del hardware y del software por el desarrollo de los protocolos, sin añadir ninguna función. Dichos costos pueden aumentar mucho si existen demasiados tipos de IED´s en la red. Por otra parte, al añadir equipos adicionales se afecta la confiabilidad global así como los requerimientos de mantenimiento a largo plazo.

Si los proveedores acordaran un estándar para el protocolo y la interfaz física del sistema de control y automatización, no se requerirían compuertas a la vez que el desarrollo de protocolos sería menos costoso. Tal como se destacó anteriormente, una red completamente integrada con su sistema automatizado necesita una red local de comunicación para unir todos los IED´s entre sí.

Los criterios envueltos en la escogencia de la red son diversos y complejos. De nuevo, así como con la interfaz de los IED´s, no existe un estándar universalmente aceptado. Sin embargo, generalmente se acepta que la Red de Área Local (LAN) tiene la topología de red apropiada.

Redes de área local. Una Red de Área Local (LAN) es típicamente muy rápida y posee un alcance hasta el patio de la subestación por lo que la transferencia de las funciones de medición, comandos de control, configuración y datos históricos entre dispositivos inteligentes en sitio es también rápida. Para toda la red eléctrica se requiere una Red de Área Amplia (WAN), que integre las LAN existentes.

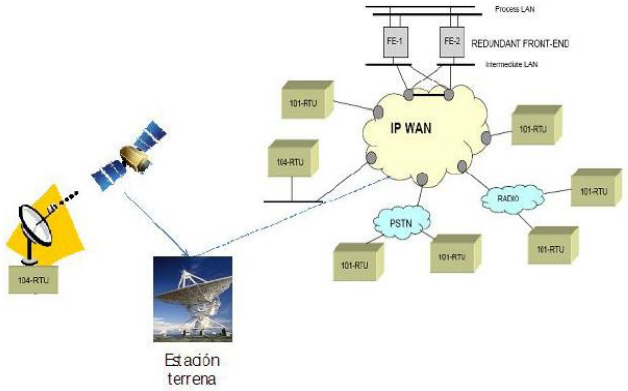


Fig. 2.8 *Red de área local.*

Esta arquitectura reduce la cantidad y complejidad del cableado requerido entre dispositivos. Más aún, incrementa el ancho de banda disponible de comunicación para realizar actualizaciones más rápidas y funciones más avanzadas tales como conexiones virtuales, transferencia de archivos, y capacidades tipo “plug and play”.

El principal punto a favor del **Ethernet** es la disponibilidad de su hardware y opciones entre una gran cantidad de proveedores, sin mencionar el apoyo del protocolo de red estándar en la industria, soporte multi-estrato y multi-aplicaciones así como calidad, y gran cantidad de equipos de prueba. Su mayor debilidad para

su uso en redes eléctricas proviene de la naturaleza no determinista del esquema de resolución empleado en su versión estándar.

Sin embargo, nuevas técnicas se han desarrollado para solucionar dicho problema. Profibus es ampliamente usado en Europa para procesos industriales y en la literatura se asegura que es determinístico, pero los protocolos de aplicación en estratos y redes están actualmente limitados a los definidos por el estándar Profibus, a la vez que las opciones existentes de hardware y equipos de prueba son inferiores en número a las ofrecidas por Ethernet.

Una vez que se hayan resuelto todos los asuntos referentes al hardware de IED´s, tecnologías LAN, y protocolos LAN e IED´s, la siguiente interrogante será cómo mostrar en pantalla o monitorear toda esta información integrada al operador de la red.

Ethernet es una tecnología de redes usada en muchas oficinas y hogares que tiene como objetivo habilitar la comunicación entre computadoras y compartir recursos, ya sea en una red local o a través de Internet. Por muchos años esta tecnología estuvo disponible sólo para computadoras, y los sistemas de monitoreo fueron limitados a interfaces de baja velocidad, rango limitado o protocolos de aplicación normalizada.

Hoy día, nuevos desarrollos en la tecnología y el mercado hacen posible la comunicación de cualquier sistema en redes locales e Internet, haciendo los sistemas de monitoreo más poderosos y sencillos de controlar.

IP e Internet. Dada ya la definición de red, el siguiente paso está en comprender el concepto de Internet, que se puede entender como un grupo amorfo de computador es en diversos países de todos los continentes, que se comunican unos con otros a través del protocolo **IP** (Internet Protocol). Cada computador en Internet tiene una única dirección **IP** mediante la cual puede ser identificado.

Internet no le pertenece a nadie ni está gobernado por nadie. Simplemente es una gran colección de computadores que han acordado comunicarse de una forma estándar. Cada máquina en una red se denomina nodo. Los nodos, que son computadores completamente funcionales, se denominan hosts. Cada nodo de red tiene una dirección, es decir, una serie de bits que lo identifica de manera única.

Cada computador en una red **IP** está identificado con un único número de 4 bytes (un byte es una unidad de información compuesta por 8 bits o unos y ceros). Este se escribe normalmente en un formato como 168.176.15.11, donde cada uno de los cuatro números es un byte sin signo que puede tomar un valor en el intervalo de O a 255.

La dirección de la fuente se incluye de manera que el receptor sepa a quién debe responder. Para que los seres humanos no tengan que recordar estos números,

se desarrolló el DNS (Domain Name System) para traducirlos a nombres de hosts que los humanos pudieran recordar. Es decir no es necesario memorizar 168.176.15.11 sino dis.unal.edu.co.

2.6 Raspberry

Un SoC (system on chip) es un integrado que incorpora todos los componentes del sistema. En el caso de la Raspi, lleva un Broadcom BCM2835 que incluye: el procesador (ARM1176JZF-S), la tarjeta gráfica con aceleración gráfica 3D y de video en alta definición, 512 Mb de RAM, tarjeta de sonido estéreo y bus USB.

Raspberry Pi Foundation fue creada en 2009 tras reconocer que, si bien la informática había logrado grandes avances durante los últimos 30 años, los ordenadores modernos no proporcionaban las mismas oportunidades de aprendizaje que brindaban los ordenadores usados en los años 80. Según Eben Upton, miembro de la Fundación, esto estaba "reduciendo la llegada a las universidades de jóvenes capaces de programar.

Fig. 2.9 *Raspberry Pi.*

Decididos a hacer algo al respecto, la Fundación se puso a diseñar un ordenador muy asequible que estimulase la enseñanza de informática básica. Upton, diseñador de sistemas integrados (SoC) de Broadcom y ex académico, estaba perfectamente posicionado para entender el problema en cuestión, y disponía de los conocimientos y los contactos necesarios para desarrollar una solución.

La Raspberry Pi adopta la forma de una PCB del tamaño de una tarjeta de crédito, utiliza SoC de Broadcom y proporciona: Un procesador ARMv6 de 700 MHz, 256 MB de RAM, Una GPU 1080 p con salidas HDMI y de vídeo, Conector de audio de 3,5 mm, Conector de 26 vías con GPIO, UART, I2C y SPI, Conectores para JTAG, DSI (display LCD) y CSI (cámara), Ranura para tarjeta SD, USB.

Conexiones. Disponemos de Dos buses USB, Puerto ethernet RJ-45, Salida analógica de audio estereo por jack de 3.5 mm., Salida digital de video + audio HDMI, Salida analógica de video RCA, Pines de entrada y salida de propósito general

Consola. Es un espacio donde se envían órdenes al sistema. Estas órdenes pueden ser de cualquier tipo: desde crear una carpeta hasta ejecutar un programa

pasando por reiniciar o apagar el sistema. Si la consola es remonta (SSH), se pueden enviar órdenes a una máquina sin necesidad de estar físicamente delante de ella.

Redirección de puertos en el router Si queremos poder acceder a los servicios que Raspberry Pi está ejecutando desde Internet (un ordenador conectado a Internet fuera de la red local en la que está conectada la Raspi), es necesario redireccionar los puertos del router. En Internet existen multitud de guías explicativas para casi todos los modelos de router. Los puertos de interés, sin los cuales no puede acceder a sus servicios son: SSH: 22, HTTP (web): 80, FTP: 21, Cliente remoto de aMule: 4712, Cliente remoto de Transmission: 9091.

2.7 Motores trifásicos de inducción

El motor de inducción trifásico, también llamado motor asíncrono, es hoy día el motor eléctrico que más se utiliza en las aplicaciones industriales, sobre todo el motor con rotor de jaula de ardilla. Su funcionamiento se basa en la interacción de campos magnéticos producidos por corrientes eléctricas.

En el caso de los motores a los que hace referencia estas notas, las corrientes que circulan por el rotor son producidas por el fenómeno de inducción electromagnética, conocido comúnmente como ley de Faraday, que establece que si una espira es atravesada por un campo magnético variable en el tiempo.

Fig. 2.10 *Motor Trifásico de inducción.*

Monitoreo de motores. Los sistemas de monitoreo de motores eléctricos suelen detectar una variedad de fallas en el motor, tales como: corriente, temperatura, cortocircuitos en los bobinados del estator, rodamientos dañados, excentricidad y desbalance en el rotor, rotura de barras y anillos en el rotor, problemas de ventilación, etc.

La sensorización de temperatura y vibración ha sido una técnica utilizada desde hace décadas para el monitoreo de motores asíncronos de inducción, sin embargo, recientemente los estudios se han centrado en el monitoreo de la parte

eléctrica del motor, haciendo énfasis en el análisis de la corriente de estator. La mayoría de los estudios se han abocado al análisis del espectro de frecuencia de la corriente de estator, con el fin de determinar fallas en el motor.

Todas las técnicas de detección de fallas requieren un conocimiento previo de las medidas recolectadas, con el fin de distinguir situaciones normales de trabajo de las condiciones de operación bajo fallo. Esta exigencia, en cuanto al conocimiento del comportamiento normal de la máquina se magnifica cuando existe en ella una gran cantidad de armónicos provenientes de características constructivas o de variación de la carga a la que se encuentra sometido el motor.

Estas características especiales no deben ser confundidas con fallos existentes en el motor. Muchos métodos de monitoreo han sido propuestos para la detección de fallos en motores asíncronos de inducción. Muchas máquinas eléctricas se encuentran provistas de distintos sensores con el fin de recolectar datos relevantes a su funcionamiento.

Algunas de estas técnicas emplean sensores frágiles, de difícil instalación en el motor y de elevado coste. Sin embargo, el monitoreo de corriente de estator puede proveer las mismas informaciones sin necesidad de acceso a la parte interna del motor, ya que se trata de una técnica no invasiva, y pudiendo así reducir los costos de implementación.

3. Desarrollo

3.1 Análisis de las bases del lenguaje python

Los números enteros son aquellos números positivos o negativos que no tienen decimales (además del cero). En Python se pueden representar mediante el tipo int (de integer, entero) o el tipo long (largo). La única diferencia es que el tipo long permite almacenar números más grandes. Es aconsejable no utilizar el tipo long a menos que sea necesario, para no malgastar memoria.

El tipo int de Python se implementa a bajo nivel mediante un tipo long de C. Y dado que Python utiliza C por debajo, como C, y a diferencia de Java, el rango de los valores que puede representar depende de la plataforma. El operador de módulo no hace otra cosa que devolvernos el resto de la división entre los dos operandos.

Bucles. Mientras que los condicionales nos permiten ejecutar distintos fragmentos de código dependiendo de ciertas condiciones, los bucles nos permiten ejecutar un mismo fragmento de código un cierto número de veces, mientras se cumpla una determinada condición.

Una función es un fragmento de código con un nombre asociado que realiza una serie de tareas y devuelve un valor. A los fragmentos de código que tienen un nombre asociado y no devuelven valores se les suele llamar procedimientos. En Python no existen los procedimientos, ya que cuando el programador no

especifica un valor de retorno la función devuelve el valor None (nada), equivalente al null de Java.

Además de ayudarnos a programar y depurar dividiendo el programa en partes las funciones también permiten reutilizar código. Esto es lo que imprime el operador de iPython o la función help del lenguaje para proporcionar una ayuda sobre el uso y utilidad de las funciones. Todos los objetos pueden tener docstrings, no solo las funciones, como veremos más adelante.

3.2 Calculo en los motores

Calculo de Corriente. Las corrientes que circulan por los conductores, concurren a un “nodo” que son los aros que unen las barras físicamente, siendo la suma de las mismas igual a cero. En esquema eléctrico equivalente se muestra en la figura.

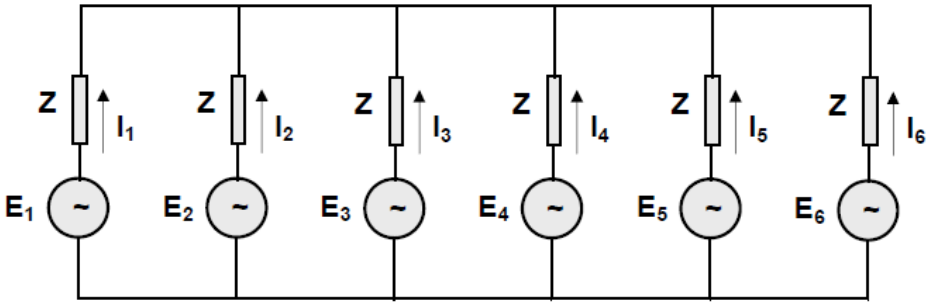


Fig. 3.1 *Esquema eléctrico equivalente.*

Las corrientes I1 e I4, tienen el mismo valor instantáneo pero el sentido de circulación es opuesto, debido a la posición que ocupan los conductores en el instante analizado. Por lo tanto las mismas tienen la misma magnitud con un ángulo de desfasaje de 180°. Lo mismo pasa con las corrientes I2 e I5 y con I3 e I6.

La construcción de este tipo de rotor es simple ya que sobre el conjunto de chapas magnéticas, en las cuales se han practicado las ranuras, se funden en las mismas las barras conductoras que por lo general son de aluminio, junto con las coronas. Esta construcción le da una gran rigidez a la jaula, ante los esfuerzos debidos a la fuerza centrifuga o bien en el caso de los esfuerzos por corrientes de cortocircuito.

Circuito equivalente del motor trifásico de inducción.- El circuito equivalente del motor trifásico de inducción o asincrónico, puede asimilarse al de un transformador. En el estator, tenemos tres bobinas que originan un campo magnético rotante. Las mismas presentan una resistencia óhmica distribuida a lo largo de los conductores que las conforman. Además parte de las líneas de campo magnético se cierran a través del aire, conformando lo que llamamos flujo disperso.

Al igual que en el transformador estos efectos los representamos por una resistencia concentrada y una reactancia de dispersión. El núcleo magnético esta dividido en dos partes, una fija que es el estator y otra móvil que es el rotor, lo cual implica una separación de aire entre ambas (entrehierro).

Por lo tanto, se va a necesitar una corriente magnetizante (bastante mayor que en el caso de un transformador debido al entrehierro mencionado), y además tenemos las perdidas en el hierro. Ambos efectos los representaremos en forma análoga, mediante una resistencia y una reactancia en paralelo. Las perdidas en el hierro del rotor son muy pequeñas, cuando gira a la velocidad de régimen, ya que las frecuencias de las corrientes son pequeñas.

Luego el estator lo podemos representar por el siguiente circuito equivalente:

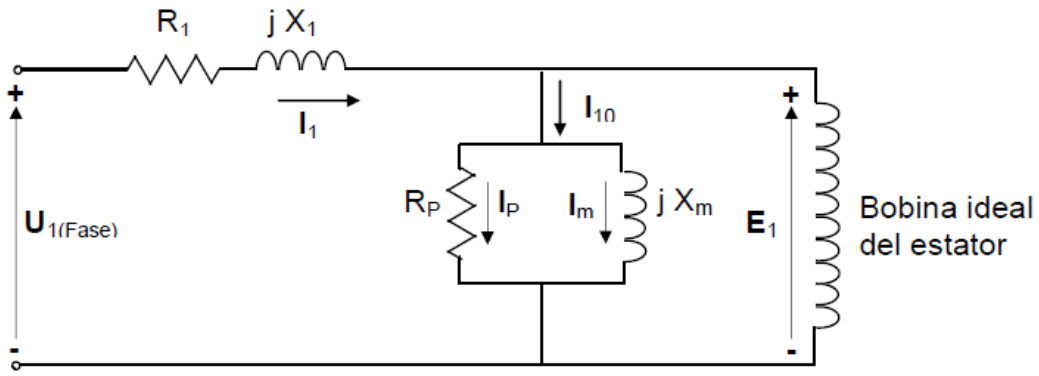


Fig. 3.2 *Circuito equivalente de una fase del estator.*

En el circuito los parámetros representan: R1: La resistencia óhmica de la bobina estatorica de una fase, X1: Reactancia de dispersión de la bobina estatorica de una fase, RP: Resistencia que representa las perdidas en el hierro por fase, Xm : Reactancia de magnetización por fase, U1: Tension de fase de alimentación al motor [V], E1: Fuerza electromotriz inducida de fase en la bobina estatorica [V], I1: Corriente estatorica con carga [A], I10: Corriente estatorica del motor en vacio [A]

En el rotor la fem inducida, dependerá de la velocidad del eje y del numero de espiras del mismo. También los conductores presentan resistencia ohmica y hay flujo disperso, el cual lo representaremos por una reactancia de dispersión cuyo valor esta dado por.

$$X2S = 2\,\pi\,fR\,L2 \qquad (3.1)$$

Siendo el valor de la autoinductancia constante, la reactancia cambia su valor con la velocidad de la maquina, como lo hace la frecuencia fR. La reactancia con el rotor detenido o bloqueado tiene el siguiente valor.

$$X2 = 2\,\pi\,f\,L2 \qquad (3.2)$$
$$X2S = 2\,\pi\,s\,f\,L2 = s\,X2 \qquad (3.3)$$

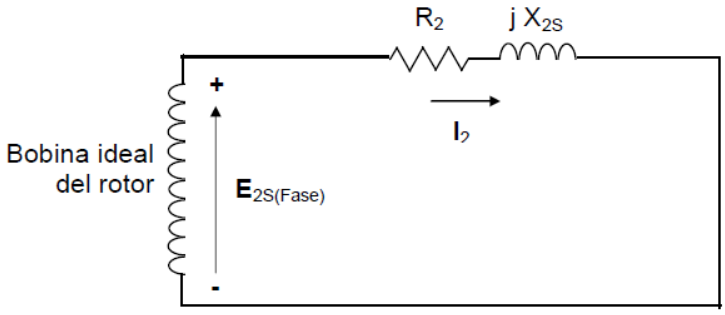


Fig. 3.3 *Circuito equivalente para una fase del rotor.*

El valor de la corriente en el rotor de acuerdo al circuito esta dada por:

$$I_2 = \frac{E_{2S}}{R_2 + j \cdot X_{2S}} \qquad (3.4)$$

Esta corriente toma valores dependientes de la fem inducida y de la reactancia, si reemplazamos:

$$E_{2S} = s \cdot E_2 \qquad X_{2S} = s \cdot X_2 \qquad (3.5)$$

$I_2 = \frac{S \cdot E_{2S}}{R_2 + j \cdot s \cdot X_{2S}}$ Dividiendo por el resbalamiento nos queda.

$$I_2 = \frac{E_2}{\frac{R_2}{s} + j \cdot X_{2S}} \qquad (3.6)$$

De esta forma nos queda un circuito equivalente en el rotor, en el cual lo variable con la velocidad es la resistencia (desde el punto de vista matemático, no físico). Teniendo los dos circuitos equivalentes del estator y del rotor, entre los mismos queda un acoplamiento inductivo, el cual lo podemos excluir, si referimos los valores del rotor al estator teniendo en cuenta la relación del numero de espiras del estator y del rotor, igual a lo que se realizo para el transformador.

En función de lo analizado el circuito equivalente por fase del motor trifásico de inducción, con sus valores referidos al estator es el de la siguiente figura.

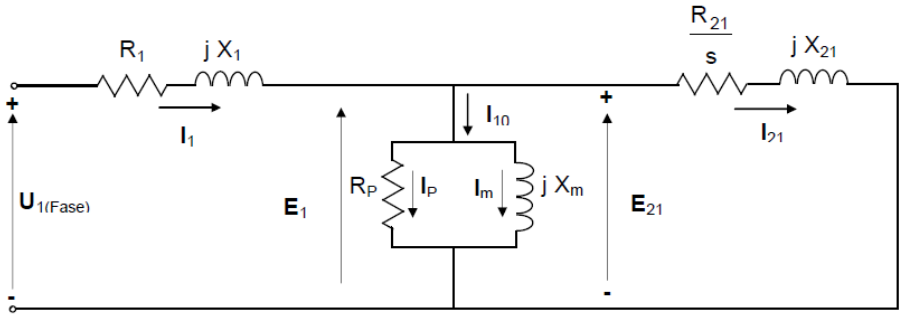


Fig. 3.4 *Circuito equivalente de una fase del motor trifásico de inducción.*

La velocidad de giro del rotor es inferior a la velocidad de giro del campo rotatorio síncrono. La diferencia entre la velocidad de giro del rotor y la velocidad de giro del campo se llama resbalamiento. El motor funciona como motor asíncrono. El **resbalamiento** suele expresarse en por ciento de la velocidad de giro del campo. Funcionando el motor con carga nominal, el resbalamiento es entre 1 % y 8 %. Cuanto menor es el resbalamiento, tanto mayor la potencia del motor.

Circuito equivalente aproximado.- A los efectos prácticos de la resolución del circuito, el mismo suele simplificarse, agrupando las perdidas en el hierro, junto con las mecánicas (no incluidas en el circuito equivalente, que solo contempla la parte electromagnética), recibiendo el conjunto el nombre de perdidas rotacionales.

$$Pr = P_{Fe} + P_m \qquad (3.7)$$

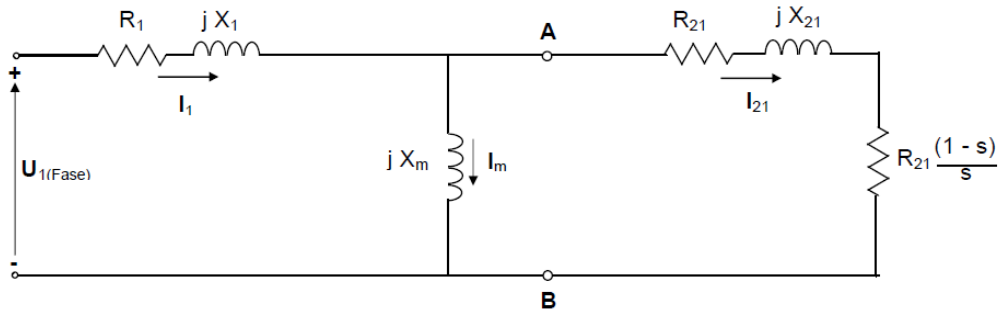


Fig. 3.5 *Circuito equivalente aproximado de una fase del motor trifásico de inducción.*

Intensidad de corriente. Si analizamos el circuito equivalente, vemos que la resistencia que representa la carga en el eje, en el momento de arranque (s = 1), toma un valor igual a cero (cortocircuito) y a medida que aumenta la velocidad va aumentando su valor hasta que en el caso de llegar a velocidad sincrónica su valor seria infinito. De aqui que la corriente varia entre el momento de arranque y la velocidad nominal o de plena carga.

La corriente que toman estos motores en el momento de arranque es elevada y del orden de 6 a 8 veces la corriente nominal, pudiéndose observar su variación en la curva de la figura.

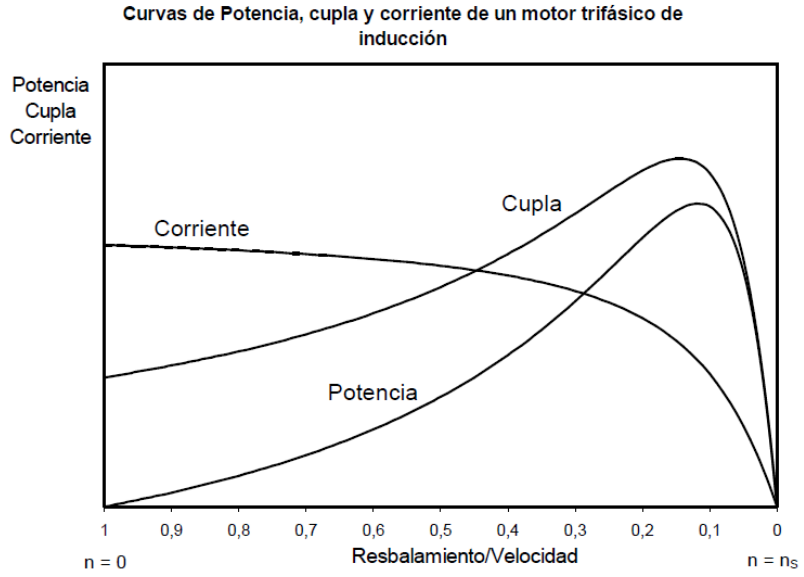


Fig. 3.6 *Variación de la potencia, cupla y corriente con la velocidad.*

3.2.3Cambio del sentido de giro de los motores monofásicos

Para invertir el sentido de giro de este tipo de motores se debe hacer que el campo rotante asi lo haga. Para ello se debe invertir la polaridad de una de las bobinas. En la figura se encuentra un esquema para efectuar el cambio.

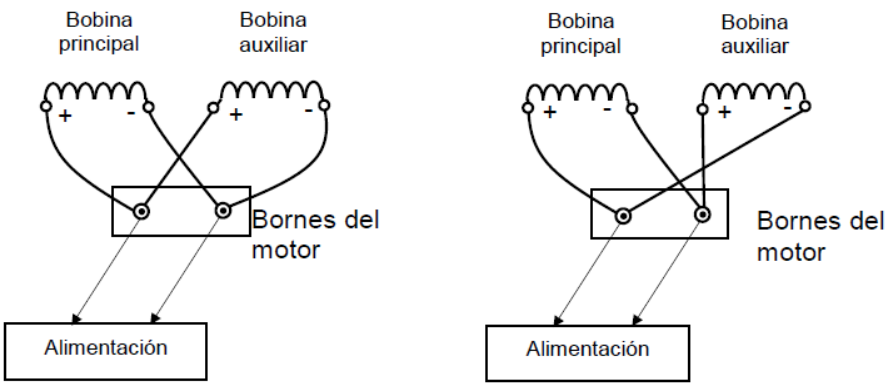


Fig. 3.7 *Esquema para realizar el cambio del sentido de giro de un motor monofásico.*

TABLA POTENCIAS NOMINALES MOTORES TRIFÁSICOS DE INDUCCIÓN
3000/1500 rpm. [Cos.φ 0,8]

KW	HP	220 V. I [A]	380 V. I [A]	660 V. I [A]
0,18	0,25	0,6	0,3	0,2
0,37	0,5	1,2	0,7	0,4
0,55	0,75	1,8	1	0,6
0,74	1	2,4	1,4	0,8
1,1	1,5	3,6	2,1	1,2
1,5	2	4,8	2,8	1,6
2,2	3	7,3	4,2	2,4
2,9	4	9,7	5,6	3,2
4	5,5	13,3	7,7	4,4
5,5	7,5	18,1	10,5	6
7,4	10	24,2	14	8,1
11	15	36,3	21	12,1
13,6	18,5	44,7	25,9	14,9
14,7	20	48,3	28	16,1
18,4	25	60,4	35	20,1
22,1	30	72,5	42	24,2
25	34	82,2	47,6	27,4
29,4	40	96,7	56	32,2
44,2	60	145	84	48,3
55,2	75	181,3	105	60,4
73,6	100	241,7	139,9	80,6
92	125	302,2	174,9	100,7
110,4	150	362,6	209,9	120,9
128,8	175	423	244,9	141
161,9	220	531,8	307,9	177,3
220,8	300	725,2	419,8	241,7

Tabla. 3.1 *Tabla potencias nominales motores trifásicos de inducción.*

Sobreintensidad de arranque. Los motores han de tener limitada la intensidad de arranque cuando esta pueda producir efectos perjudiciales para la instalación u ocasionar perturbaciones en otros receptores.

TABLA DE PROPORCIONALIDAD ENTRE CORRIENTE DE ARRANQUE Y LA DE PLENA CARGA

Motores corriente continua Potencia nominal		Motores corriente alterna Potencia nominal	
De 0,75 KW a 1,5 KW	2,5	De 0,75 KW a 1,5 KW	4,5
De 1,5 KW a 5,0 KW	2,0	De 1,5 KW a 5,0 KW	3,0
Mes de 5,0 KW	1,5	De 5,0 KW a 15,0 KW	2,0
		Mas de 15,0 KW	1,5

Tabla. 3.2 *Tabla proporcionalidad entre corriente de arranque y la de plena carga.*

3.3 Temperatura en motores

Calentamiento y ventilación. La vida útil de un motor es igual a la del aislamiento de sus devanados, si se prescinde del desgaste propio del servicio de los

cojinetes, escobillas, anillos rozantes o colector, elementos que se pueden sustituir por otros nuevos sin que, relativamente, se realicen gastos de importancia. Por este motivo, se tendrán especialmente en cuenta las condiciones de servicio que afecten al calentamiento y, por tanto, al aislamiento.

El calentamiento es una consecuencia de las pérdidas originadas en toda transformación de energía (en caso de motores, por ejemplo, transformación de energía eléctrica en energía mecánica).

El calentamiento del motor se produce, principalmente, por las pérdidas en el hierro de las chapas magnéticas y del núcleo y por las pérdidas en el cobre del devanado. Estas últimas calientan también el aislamiento de cada conductor. La temperatura admisible del aislamiento utilizado determina fundamentalmente la capacidad de carga del motor.

$$\text{Ppérd.} = \text{Pabs.} - \text{Pced} \qquad \textit{(3.8)}$$

En la práctica no se indican las pérdidas del motor, sino su rendimiento, el cual se calcula de la siguiente forma

$$\eta = \frac{Pced}{Pced + Pp\acute{e}rd} \cdot 100 \qquad \textit{(3.9)}$$

Siendo; Ppérd. = pérdidas totales (kW), Pced. = potencia (kW), Pabs. = potencia activa (kW) que se entrega en el eje tomada de la red, η = rendimiento (%)

La energía consumida en pérdidas = pérdidas por tiempo en kWh (calor), se acumula en el motor, de acuerdo a su capacidad térmica, conduciéndose una gran parte al medio ambiente, a través de la ventilación. Si la carga es constante, se alcanzará un estado de equilibrio cuando la cantidad de calor absorbida sea igual a la disipada, en servicio continuo, una vez que hayan transcurrido de 3 a 5 horas.

La sobretemperatura entonces motivada (calentamiento) en los devanados y en el resto de las partes del motor es igual a la diferencia que hay entre la temperatura de la parte considerada y la del medio refrigerante. La sobretemperatura resulta de la relación existente entre las pérdidas que en el motor se transforman en calor y la capacidad de disipación del calor.

$$ST = \frac{Pp\acute{e}rd}{Wa} \qquad \textit{(3.10)}$$

Siendo; ST = sobretemperatura (°C), Ppérd. = pérdidas (W), Wa = capacidad de disipación del calor (W / °C)

La capacidad de disipación de calor depende de la superficie exterior del motor y de las condiciones de ventilación. Como la duración del aislamiento de los devanados decrece al aumentar la temperatura (cada 10 °C, aproximadamente en la mitad), según sea el material utilizado habrá que observar los valores límites fijados por VDE 0530 para la temperatura del devanado (temperatura límite).

Estos valores están de acuerdo con la respectiva resistencia térmica de los materiales aislantes subdivididos en clases. La duración media prevista es, aproximadamente, de 20 años.

Aumento de temperatura. Estándar de máxima temperatura permisible del ambiente: 40 ºC. Temperatura ambiente (Estándar AIEE No. 1, 1947) : es la temperatura del medio empleado para enfriamiento, directo o indirecto, esta temperatura se resta de la temperatura medida en la máquina para determinar el aumento de temperatura bajo condiciones específicas de prueba. El aumento máximo permisible de temperatura es sobre éste estándar de 40 ºC.

Limites de Temperatura

Factor de servicio: 1		
Clase	Aumento máximo permisible sobre 40 °C	Temperatura máxima del punto más caliente
B	80	120
F	105	145

Factor de servicio: 1.15		
Clase	Aumento máximo permisible sobre 40 °C	Temperatura máxima del punto más caliente
B	90	130
F	115	155

Tabla. 3.3 *Limites de Temperatura*

Temperatura de la carcasa. De acuerdo a las técnicas constructivas modernas, y tomando en cuenta las normas sobre materiales aislantes y clases de aislamiento, los fabricantes de motores utilizan la particularidad de unir lo más cerca posible el paquete del estator a la carcasa, de manera que se evacue rápida y eficientemente el calor interno generado por las diferentes partes constitutivas del motor. Es por esto que el método antiguamente utilizado, para determinar si un motor está sobrecargado o no, tocando con la mano la carcasa, es completamente inadecuado para motores eléctricos modernos.

Calentamiento del local. El calentamiento del local depende exclusivamente de las pérdidas, y no de la temperatura de la carcasa. Además, las máquinas accionadas frecuentemente contribuyen al calentamiento del local en mayor proporción que los motores. En todas las máquinas elaboradoras y modificadoras de materiales, se transforma prácticamente la totalidad de la potencia y accionamiento en calor, y en las máquinas transportadoras de material la transformación se extiende a una gran parte de la potencia de accionamiento.

Estas cantidades de calor tienen que ser eliminadas por el aire ambiental en el local de servicio, a no ser que los motores tengan refrigeración independiente, consistente en un sistema de tubos a través de los cuales se evacua el calor directamente al exterior. Habrá que considerar lo siguiente.

$$V_L = \frac{\text{Ppérd} \cdot 0.77}{J} \qquad (3.11)$$

V_L = caudal de aire necesario (m3/s), Ppérd.= potencia total de pérdidas (kW), J = sobretemperatura admisible del aire (°C)

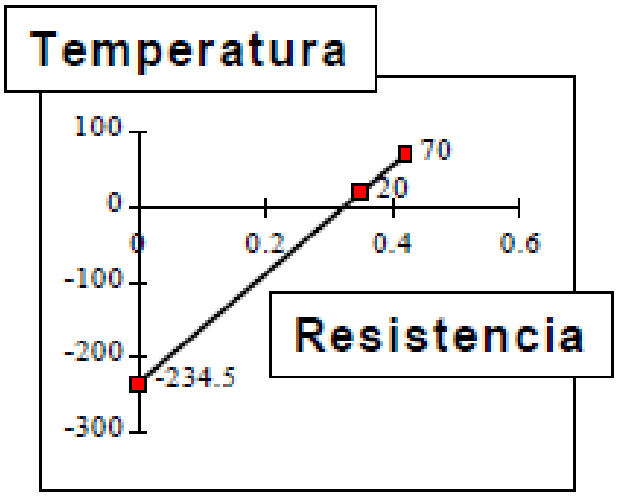
$$LW/_h = \frac{V_{Lu}}{Jv} \qquad (3.12)$$

LW / h= número de renovaciones de aire por hora, V_{Lu} = caudal de aire en circulación (m3/h), J_V = volumen del local (m3)

Durante el servicio hay que conseguir un buen abastecimiento de aire fresco para refrigerar los motores. Los motores de gran tamaño provistos de refrigeración interna necesitan un caudal horario de aire que es, aproximadamente, 4 ó 5 veces mayor que su peso propio (a 760 Torr y 20 °C, 1 m3 de aire pesa 1,2 kg). Un motor de 120 kW y 1.800 rpm provisto de refrigeración interna necesita en una hora 2.000 m3 de aire.

Refrigeración y ventilación. Todos los motores tienen un ventilador exterior cubierto con una caperuza. Independientemente del sentido de giro del motor, dicho ventilador impulsa el aire de refrigeración sobre la superficie. El ventilador y su caperuza correspondiente están conformados para que la corriente de aire refrigerante no pueda acumular suciedad ni fibras que podrían obstaculizar la refrigeración.

Calculo de temperatura del punto más caliente.- En todas las máquinas elaboradoras y modificadoras de materiales, se transforma prácticamente la totalidad de la potencia y accionamiento en calor, y en las máquinas transportadoras de material la transformación se extiende a una gran parte de la potencia de accionamiento**.** Se calcula la temperatura del punto mas caliente del motor de acuerdo a lo siguiente.



$$\frac{T_2+234.5}{R_2} = \frac{T_1+234.5}{R_1} \qquad (3.13)$$

$$T_2 = \frac{R_2}{R_1}(T_1 + 234.5) - 234.5 \qquad (3.14)$$

R_1=Resistencia medida en frio (T_1< 40°C)
R_2=Resistencia medida inmediatamente después de operación prolongada.

$$T_{punto\ \ mas\ \ caliente} = T_2 + 10°C \qquad (3.15)$$

Arranque y giro del motor.- Con independencia del arranque directo, el arrancador estrella-triángulo es el sistema de arranque más utilizado en los motores asíncronos de inducción. Consiste en arrancar el motor con conexión estrella a una tensión √3 veces inferior a la que soporta el motor para este tipo de conexión, transcurrido un cierto tiempo, cuando el momento desarrollado por el motor conectado en estrella Mλ iguale al momento de la carga (alrededor del 80% de la velocidad nominal) conmutar las conexiones de bobinas del motor a triángulo.

Los esquemas de la automatización del arranque mediante contactores de un arrancador estrella-triángulo son los que se indican en la siguiente figura.

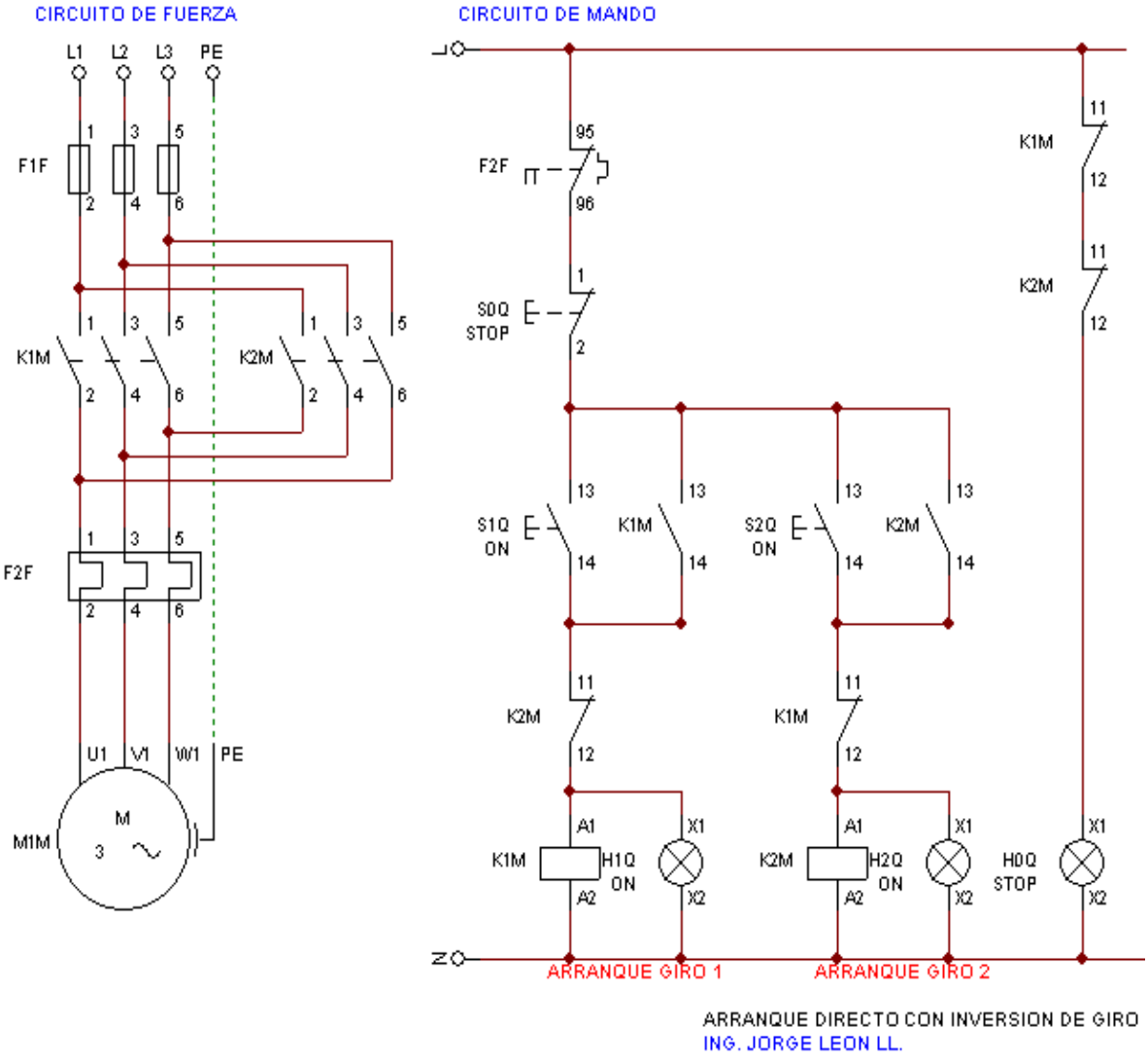


Fig. 3.8 *Circuito de potencia y control.*

3.4 Diseño de interface en Python- Glade

El diseño e implementación de la interface se da conforme el usuario requiere en este caso a lo que especifica el objetivo, en la siguiente imagen se observa como se ejecuta nuestro programa desde python, y automáticamente habre el .exe de glade. Esta fig. 3.9. Es en sí el programa ya en función desde una computadora portátil. Así mismo se ira detallando la realización del sistema completo.

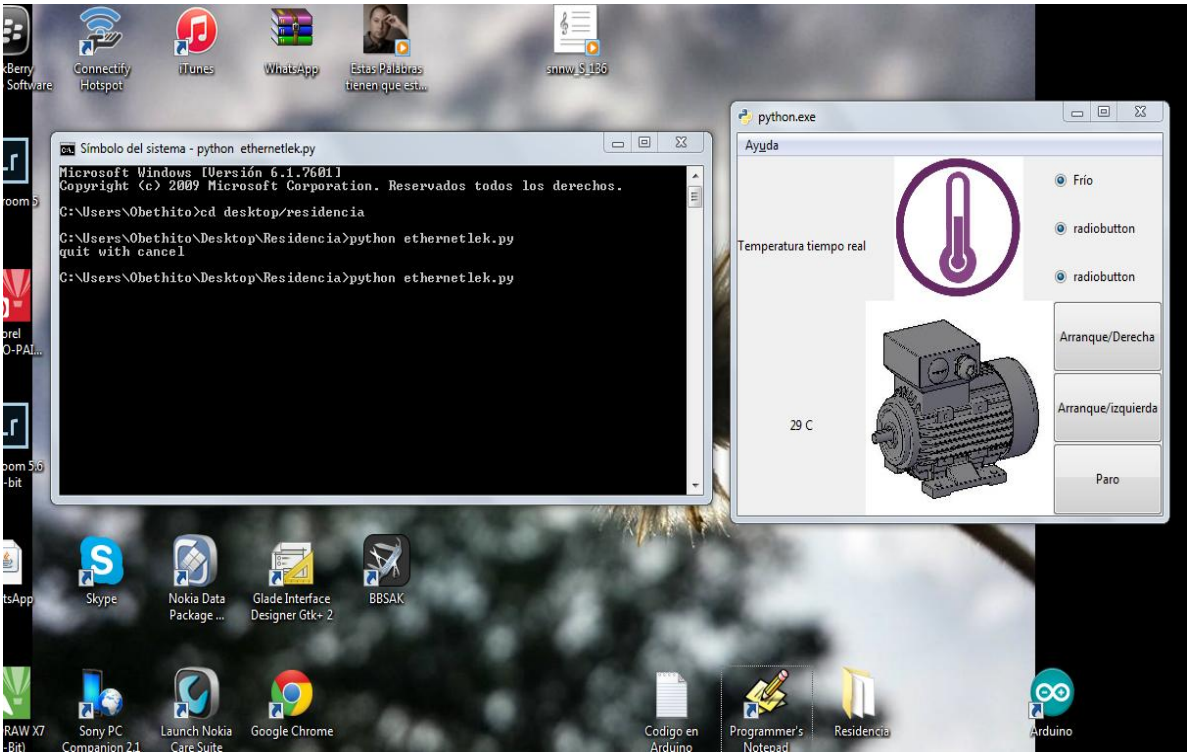


Fig. 3.9 *Programa en función.*

Programación en Python.- Por lo general, el intérprete de Python se instala en file:*/usr/local/bin/python* en las máquinas dónde está disponible; poner /usr/local/bin en el camino de búsqueda de tu intérprete de comandos Unix hace posible iniciarlo ingresando la orden: python En máquinas con Windows, la instalación de Python por lo general se encuentra en C:\Python27, aunque se puede cambiar durante la instalación. Enseguida se da comienzo con la programación.

```
import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host ="169.254.150.63"
port =80
s.connect((host,port))
```

Fig. 3.10 *Importación de librería Ethernet.*

Se comienza importando las librerías que se van a utilizar, para hacer la programación más limpia e intuitiva se importa la librería que nos ayudará a hacer la comunicación vía Ethernet (Fig. 3.10) con el dispositivo secundario, en este caso un Arduino Ethernet. Se declara la variable s para hacer el código menos extenso y declaramos la dirección ip y el puerto a utilizar, posteriormente se realiza la conexión.

```
import gtk
```

Fig. 3.11 *Importación gtk.*

Se importa la librería gtk (Fig. 3.11) que sirve para reconocer el archivo generado en glade en el cual se realiza el diseño de la interfaz. Esta librería trabaja – en este caso- con la versión Gtk +2. Esta versión nos deja utilizar una serie de elementos de una forma más sencilla.

```
class Ejemplo:
```

Fig. 3.12 *Clases.*

La programación en Python es más limpia que otras plataformas de programación, es esta se usan las clases las cuales se definen con la clave class (fig. 3.12) seguida del nombre de la clase y dos puntos; a continuación el cuerpo de la clase.

```
def on_window1_destroy(self, object, data=None):
  print "quit with cancel"
  gtk.main_quit()

def on_botonarranquede_clicked(self, object, data=None):
  print "arranque/derecha"

  while True:
          data = connsocket.recv(1024)
          connsocket.send("LED=D")

  connsecket.close()

def on_botonparo_clicked(self, object, data=None):
  print "paro"
  while 1:
   (clientsocket, address) = s.accept()
   print ("connection found!")
   data = clientsocket.recv(1024).decode()
   print (data)
   r='LED=P'
   clientsocket.send(r.encode())


def on_botonarranqueizq_clicked(self, object, data=None):
  print "arranque/izquierda"
```

Fig. 3.13 *Definición de variables.*

Posteriormente se declaran las funciones (fig. 3.13). Se declaran con la clave def seguida del nombre de la función, para este caso se declaran tantas funciones como son necesarias. Se usan para los botones, para etiquetas de entrada y acciones.

```
while 1:
 (clientsocket, address) = s.accept()
 print ("connection found!")
 data = clientsocket.recv(1024).decode()
 print (data)
 r='LED=P'
 clientsocket.send(r.encode())
```

Fig. 3.14 *Envío de caracteres por Ethernet.*

Se utiliza el bucle while para ejecutar el código de envío de datos a Ethernet, dado que en Arduino se declara el string para realizar una acción, aquí se ejecuta el while mientras se ejecuta dicha condición. Para el caso del botón ‘paro’ se envía LED=P (fig. 3.14).

```
while 1:
   e = arduino.readline()
   temp= e [0:4]
readingtempe.set(temp)
readingtempe = StringVar()
lbl4= Label(window1,textvariable = readingtempe)
```

Fig. 3.15 *Recepción de datos desde Ethernet.*

Un nuevo while para la recepción de datos (fig. 3.15). Este while se usa para recibir la información desde Arduino de la temperatura y de entrada analógica de la corriente. El código que sigue es para determinar el lugar de impresión dentro de la interfaz.

```
def __init__(self):
  self.gladefile = "motorcontrol.glade"
  self.builder = gtk.Builder()
  self.builder.add_from_file(self.gladefile)
  self.builder.connect_signals(self)
  self.window = self.builder.get_object("window1")
  self.window.show()
```

Fig. 3.16 *Iniciación de la interfaz desde glade.*

La última función que se define en la clase es init (fig. 3.16). Esta sirve para iniciar la interfaz construida en glade, la cual se integra en la ventana principal (window1). Por último se determina la clave show para abrirla ventana.

```
if __name__ == "__main__":
  main = Ejemplo()
  gtk.main()
  arduino.close()
```

Fig. 3.17 *Iniciación del programa*

Al final de la programación se ejecuta el programa y se cierran las variables ejecutadas durante la iniciación del programa: la clase, el archivo generado en glade el cual trabaja bajo Gtk y la comunicación con Arduino.

Diseño en Glade.- En la opción *Archivo* de la Ventana principal elegimos *Nuevo Proyecto* y, lógicamente, respondemos afirmativamente a la pregunta de Glade sobre si realmente deseamos crearlo. Se da comienzo al diseño conforme a lo requerido. El diseño que se realiza en glade es con los datos que nos interesa del motor trifásico para que tenga un funcionamiento correcto.

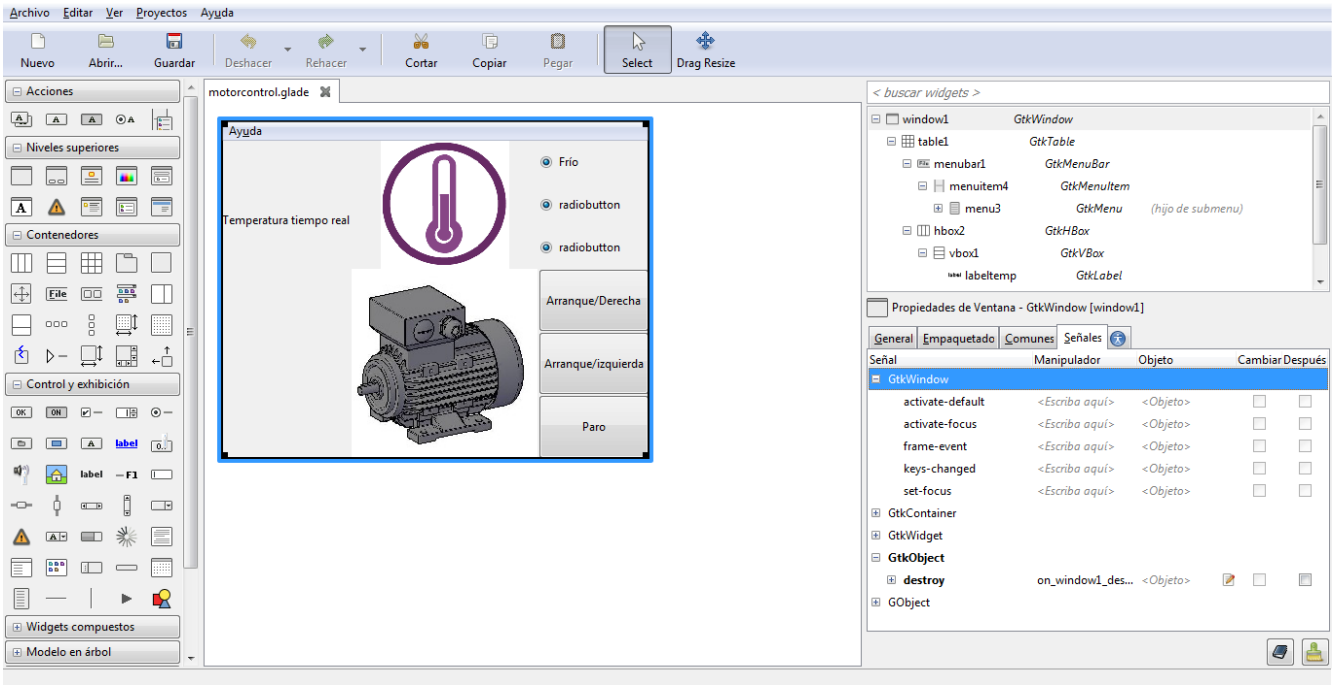


Fig. 3.18 *Diseño en glade (propiedades de ventana.*

Glade se usa como una herramienta que nos permite el desarrollo de la interfaz del programa. Glade genera un archivo en XML en el cual también se pueden hacer modificaciones y mediante el uso de Gtk se carga para la aplicación. Mediante GtkBuilder, el archivo generado en XML Glade se utiliza en e lenguaje de programación Python.

Como se observa en la figura 3.18 la manera a usar Galde queda a decisión del usuario. Para este programa utilizamos botones que nos sirven para la manipulación del motor, etiquetas que nos sirven para la impresión de los datos obtenidos en Arduino y los selectores que nos indican la temperatura a la que se encuentra el motor.

Programación en Arduino.- Primero se seleccionan las librerías que se usarán, para la comunicación Ethernet se usa la librería Ethernet y SPI. Se escribe la dirección mac e ip en la cual se hará la comunicación (fig. 3.19). Se declaran los pines de salida y entrada. Se leen los caracteres de una secuencia de caracteres y se determinan la cantidad máxima de los mismos.

En la iniciación del programa se determina que el motor estará apagado y que mientras la temperatura no sea estable (menor a la temperatura determinada como máxima) el motor permanecerá apagado.

Si la temperatura es estable, se consideran varias condiciones 'if' las cuales nos ayudan a determinar la función de cada botón en la interfaz del programa. Se leen los caracteres con 'readString' y se determina el tipo de string que se maneja durante la ejecución de los botones. Para la interfaz se envían los datos con 'print' sujeto al cliente ya definido. Estos datos son temperatura y corriente.

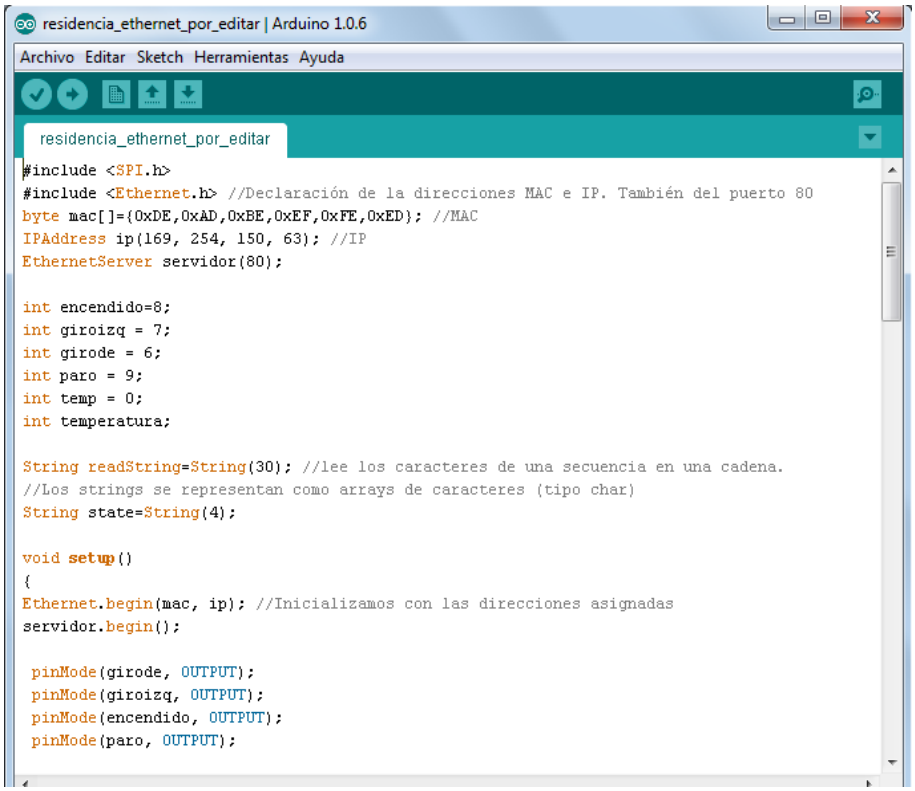


Fig. 3.19 *Programación en arduino(Completo en Anexos).*

3.5 Conexiones

Las conexiones se inician desde arduino, sensores, relevadores, contactores e interruptores. En la figura 3.20 se muestra la conexión del arduino a los sensores

que están colocados en el motor conectados a prueba en un protoboard, de igual forma la conexión de los contactores.

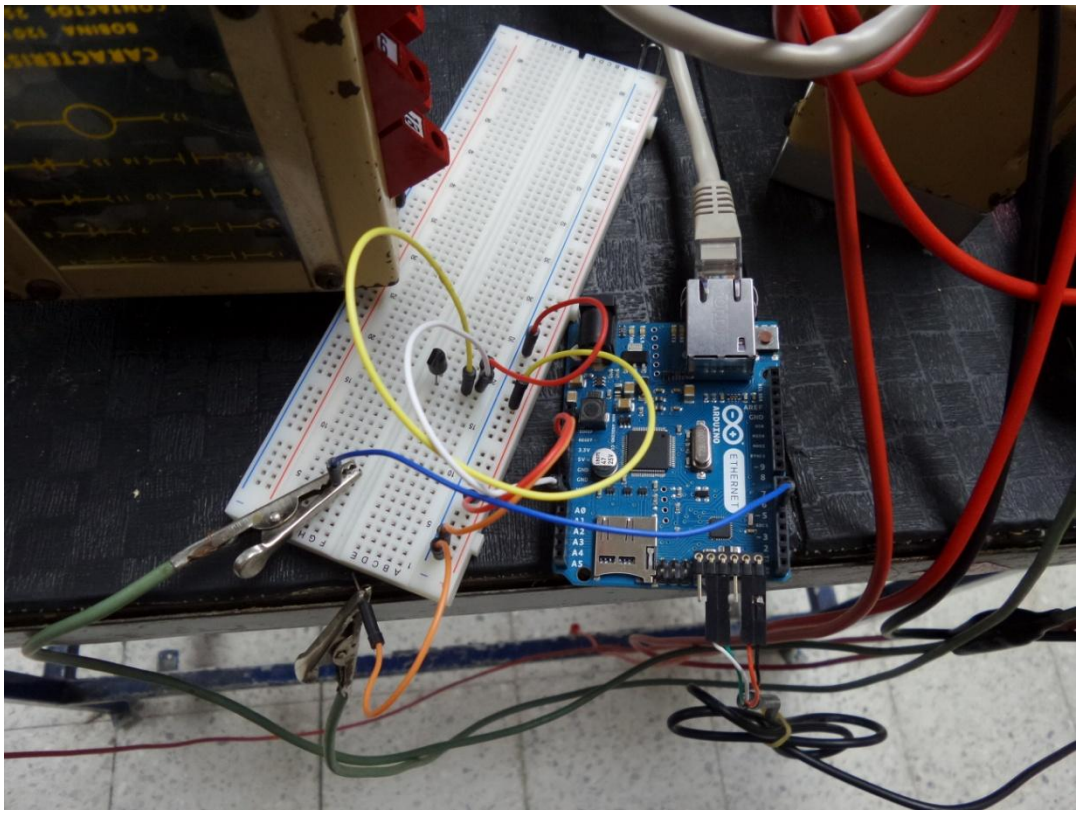


Fig. 3.20 *Conexión arduino- protoboard y sensores.*

En la figura 3.21 se muestra la conexión de los diferentes componentes que hacen que nuestro sistema realice el control y el monitoreo del motor trifásico, en la figura se aprecia el motor de laboratorio, esto se hace para llevar acabo las conexiones de manera experimental, juntamente con los contactores y relevadores.

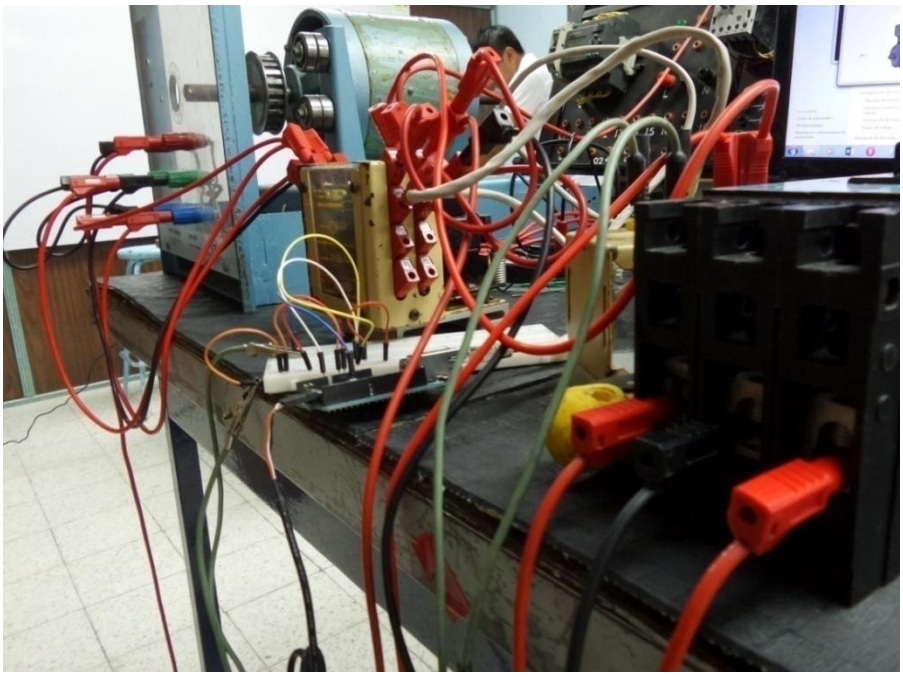


Fig. 3.21 *Conexión de todo el sistema.*

4. Resultados y conclusiones

4.1 Resultados

En la siguiente figura se ve como el circuito queda conectado y funcionando, todos los componentes funcionan correctamente, se comprueba que todo esta conectado debidamente, usando los cables adecuados para los equipos y componentes que usamos. La conexión quedo de manera satisfactoria haciendo que el sistema quede correctamente.

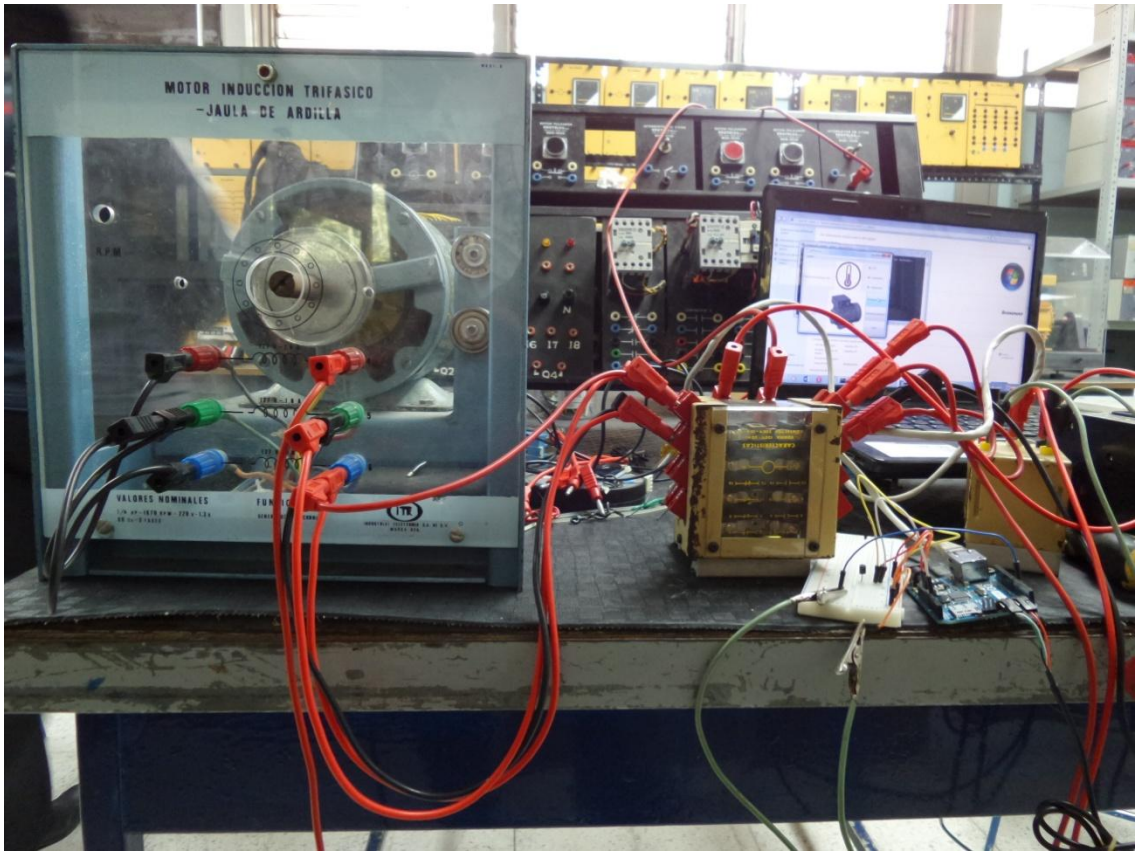


Fig. 4.1 Sistema funcionando.

En la figura 4.2 se muestra el sistema completo funcionando correctamente, así mismo ya monitoreando desde una computadora la temperatura del motor, realizando la función que debe hacerse de acuerdo a la programación de los

diferentes tipos de software haciendo que el motor siga funcionando de manera correcta.

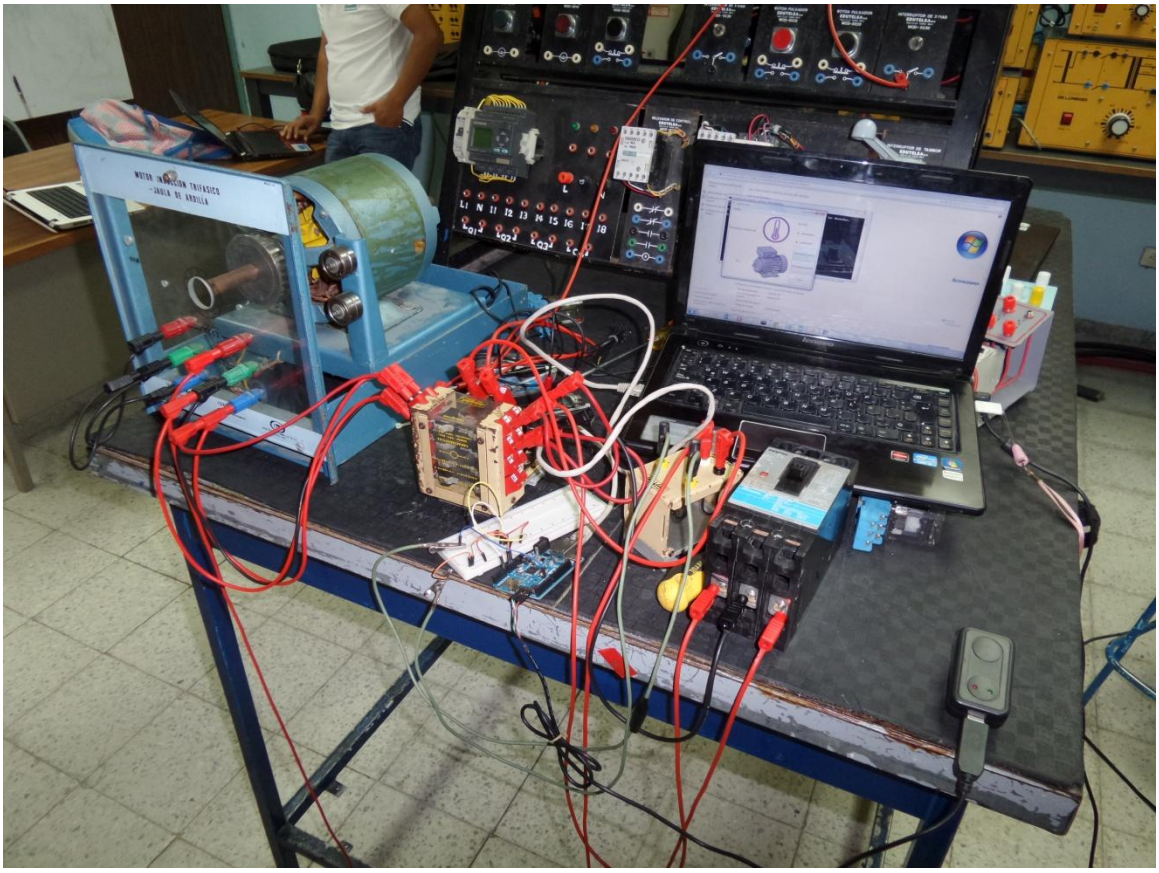


Fig. 4.2 Control de temperatura desde la computadora.

4.2 Conclusiones

Se desarrolla esta interface de manera satisfactoria, obteniendo los resultados correctos, logrando un monitoreo a los motores con los resultados de temperatura y corriente con mayor facilidad y rapidez que los diferentes métodos, de igual forma se logra el control de estos como encendido y apagado de manera automática o de manera manual conforme a la temperatura y la corriente, así mismo cuidando el equipo de cualquier riesgo que pueda ocurrir.

Al utilizar el Ethernet para llevar acabo el monitoreo y control es un avance en la tecnología ya que se puede monitorear desde diferentes puntos y así mismo tener una amplia oportunidad en las empresas que usan estos equipos.

Referencias Bibliográficas

[1] Guía de aprendizaje de Python, *Release 2.0*. Guido van Rossum. Fred L. Drake, Jr., editor; 16 de octubre de 2000, BeOpen PythonLabs, python-docs@python.org

[2] Castelli, M. Andrade, M. Portillo, A. Daoudian, N. Farinella, D. Casas, N. Desarrollo de un equipo para realización de mantenimiento predictivo en motores asíncronos de gran porte. URUMAN 2006. 16-18.

[3] Alfaro, V.M. (2011). Sistemas de control realimentado. Presentación Powerpoint para el curso *Sistemas de Control.* Escuela de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de Costa Rica.

[4] García Padilla Rubén, Aplicación Android para Supermercados, Tesis de Titulación en Ingeniería Informática Técnica de Gestión, Facultad de Informática de Barcelona, Barcelona-España, 2011.

[5] Carrillo Paz, A. J. (2009). Modelo Didáctico para el Aprendizaje Significativo en los Sistemas Automáticos de Control. Tomado de: http://www.publicaciones.urbe.edu/index.php/REDHECS/article/viewArticle/615/1563

[6] Arroyo González, Abdiel y Duran Balderas, Erick Martin 2010. Desarrollo de tecnología Ethernet y sus aplicaciones. Universidad Autónoma de Querétaro.

[7] Bernspang, Johan 2004. Interfacing an external Ethernet MAC/PHY to a MicroBlaze system on a Virtex-II FPGA. Universidad de Queensland.

[8] Guzmán Gallegos, Ricardo 2009. Monitoreo y control de riego mediante una red Ethernet con interfaz en labview: Adquisición de datos en una red modbus plus a través de una red interbus. Universidad Autónoma de Querétaro, Facultad de Ingeniería.

[9] López Gaudencio, Silvia 2010. Monitoreo y control a distancia de un interruptor de distribución mediante el microcontrolador MSP430 de Texas Instruments. Universidad Tecnológica de la Mixteca.

[10] Macías Fernández, Ricardo 2007. Sistema en tiempo real para las estaciones de monitoreo SISMO1. Universidad de Colima.

[11] Martínez Velázquez, Gabriela 2009. Comunicación de elemento final de control con red Ethernet. Universidad Autónoma de Querétaro, Facultad de Ingeniería.

[12] Méndez Bautista, David 2009. Sistema de comunicaciones basado en Ethernet para el control de sistemas empotrados. Universidad Tecnológica de la Mixteca.

[13] Mohsenin, Tinoosh 2004. Desing and evaluation of FPGA-Based Gigabit-Ethernet/PCI Network Interface Card. Universidad Rice.

[14] Toledano Ayala, Manuel 2006. Diseño e implementación de un sistema de monitoreo remoto para un invernadero. Universidad Autónoma de Querétaro, Facultad de Ingeniería.

[15] Toledano Ayala, Manuel 2010. Arquitectura hibrida basada en sistemas embebidos para el monitoreo remoto de largo alcance de estaciones de sensores. Universidad Autónoma de Querétaro, Facultad de Ingeniería.

[16] Castelli, M. Andrade, M. Portillo, A. Daoudian, N. Farinella, D. Casas, N. Desarrollo de un equipo para realización de mantenimiento predictivo en motores asíncronos de gran porte. URUMAN 2006. 16-18, Agosto 2006, Montevideo – Uruguay.

[17] TOMASI, Wayne. Sistemas de Comunicaciones electrónicas. Englewood Cliffs N.J. prentice-hall, 1996.

[18] William Stallings. Comunicaciones y Redes de Computadores, 68 edición, Prentice-Hall Internacional, 2000.

[19] Kuo, Benjamín C. “Sistemas de Control Automático”, Ed.7, Prentice Hall, 1996, México.

[20] ARDUINO - Homepage, (última modi_cación, Julio 2008). Disponible en: http://www.arduino.cc

[21] http://blog.bricogeek.com/noticias/arduino/duinos---sistema---operativo--multitarea---para---arduino/

[22] JIMENEZ MACIAS, Emilio. Técnicas de automatización avanzadas en procesos industriales (Tesis Doctoral). UNIVERSAD DE LA RIOJA, 2001.

[23] Catalogo interfaz grafica de operador PM850. SCHNEIDER ELECTRIC. 2009. Pag. 156.

[24] http://glade.gnome.org

[25] http://openmicros.org/index.php/articles/94-ciseco-product documentation/raspberry-pi/217-getting-started-with-raspberry-pi-gpio-and-python

[26] http://raspberrypihelp.net/tutorials/29-raspberry-pi-no-ip-tutorial

[27] http://www.app.etsit.upm.es/departamentos/teat/asignaturas/labingel/motor%20asincrono%20trifasico.pdf

[28] Python para todos por Raúl González Duque

ANEXOS

Programación en C del arduino.

```
#include<SPI.h>
#include<Ethernet.h> //Declaración de la direcciones MAC e
IP. También del puerto 80
bytemac[]={0xDE,0xAD,0xBE,0xEF,0xFE,0xED}; //MAC
IPAddress ip(169, 254, 150, 63); //IP
EthernetServer servidor(80);

int encendido=8;
int giroizq = 7;
int girode = 6;
int paro = 9;
int temp = 0;
int temperatura;

StringreadString=String(30); //lee los caracteres de una
secuencia en una cadena.
//Los strings se representan como arrays de caracteres (tipo
char)
String state=String(4);

void setup()
{
Ethernet.begin(mac, ip); //Inicializamos con las direcciones
asignadas
servidor.begin();

 pinMode(girode, OUTPUT);
 pinMode(giroizq, OUTPUT);
 pinMode(encendido, OUTPUT);
pinMode(paro, OUTPUT);


digitalWrite(encendido, LOW);
```

```
digitalWrite(girode, LOW);
digitalWrite(giroizq, LOW);
digitalWrite(paro, HIGH);

state="Paro";
}
voidloop()
{
  //EthernetClient Crea un cliente que se puede conectar a
  //una dirección específica de Internet IP
EthernetClient cliente= servidor.available();
 if(cliente)
 {
  boolean lineaenblanco=true;
  while(cliente.connected())
  {
   if(cliente.available())
   {
    char c=cliente.read();
    if(readString.length()<30)
    {
      readString.concat(c);
//Cliente conectado
      //Leemos petición HTTP caracter a caracter
      //Almacenar los caracteres en la variable readString
     }
if(c=='\n' &&lineaenblanco) //Si la petición HTTP ha
finalizado
     {
temperatura = analogRead(temp);
temperatura = 5.0*temperatura*100/1024;
delay(1000);

  if(temperatura >=30){
    digitalWrite(paro, HIGH);
    digitalWrite(girode, LOW);
    digitalWrite(giroizq, LOW);
    digitalWrite(encendido, LOW);
    state="Paro";
  }
      int LED = readString.indexOf("LED=");
      if(readString.substring(LED,LED+5)=="LED=D")
{
digitalWrite(paro, LOW);
digitalWrite(girode, LOW);
digitalWrite(encendido, HIGH);
delay(15000);
```

```
digitalWrite(giroizq, HIGH);
digitalWrite(paro, LOW);
digitalWrite(encendido, HIGH);
state="Arranque/Derecha";
}
      else if (readString.substring(LED,LED+5)=="LED=P")
        {
        digitalWrite(giroizq, LOW);
        digitalWrite(girode, LOW);
digitalWrite(paro, HIGH);
digitalWrite(encendido, LOW);
state="Paro";
        }
         else if (readString.substring(LED,LED+5)=="LED=I")
{
digitalWrite(paro, LOW);
digitalWrite(giroizq, LOW);
digitalWrite(encendido, HIGH);
delay(15000);
digitalWrite(girode, HIGH);
digitalWrite(paro, LOW);
digitalWrite(encendido, HIGH);
state="Arranque/Izquierda";
        }
cliente.print(temperatura);
cliente.stop();
//Cierro conexión con el cliente
readString="";
}
}
}
}
}
```

Programación en Python

```
#!/usr/bin/env python

import socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host ="169.254.150.63"
port =80
s.connect((host,port))

import gtk

class Ejemplo:

  def on_window1_destroy(self, object, data=None):
```

```
    print "quit with cancel"
    gtk.main_quit()

  def on_botonarranquede_clicked(self, object, data=None):
    print "arranque/derecha"

    while True:
            data = connsocket.recv(1024)
            connsocket.send("LED=D")

    connsecket.close()

  def on_botonparo_clicked(self, object, data=None):
    print "paro"
    while 1:
     (clientsocket, address) = s.accept()
print ("connection found!")
     data = clientsocket.recv(1024).decode()
     print (data)
     r='LED=P'
     clientsocket.send(r.encode())


  def on_botonarranqueizq_clicked(self, object, data=None):
    print "arranque/izquierda"


  def etiquetatemperatura(self, object, data=None):
     while 1:
           e = arduino.readline()
           temp= e [0:4]
     readingtempe.set(temp)
     readingtempe = StringVar()
     lbl4= Label(window1,textvariable = readingtempe)

  def on_frio_group_changed(self, object):
     arduino.readline()
     if ('<30'):
        print 'frio'

  def __init__(self):
    self.gladefile = "motorcontrol.glade"
    self.builder = gtk.Builder()
    self.builder.add_from_file(self.gladefile)
    self.builder.connect_signals(self)
    self.window = self.builder.get_object("window1")
    self.window.show()

if __name__ == "__main__":
main = Ejemplo()
  gtk.main()
  arduino.close()
```