CUADRO DE PROTECCIONES CONTROLADO MEDIANTE MICROCONTROLADOR ARDUINO, GOBERNADO Y MONITORIZADO DESDE DISPOSITIVO ANDROID

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

ANEXOS:

ÍNDICE.

INDICE DE FIGURAS.

1.- ANEXO 1 RELES OPTOACOPLADOS

1.1.- Descripción

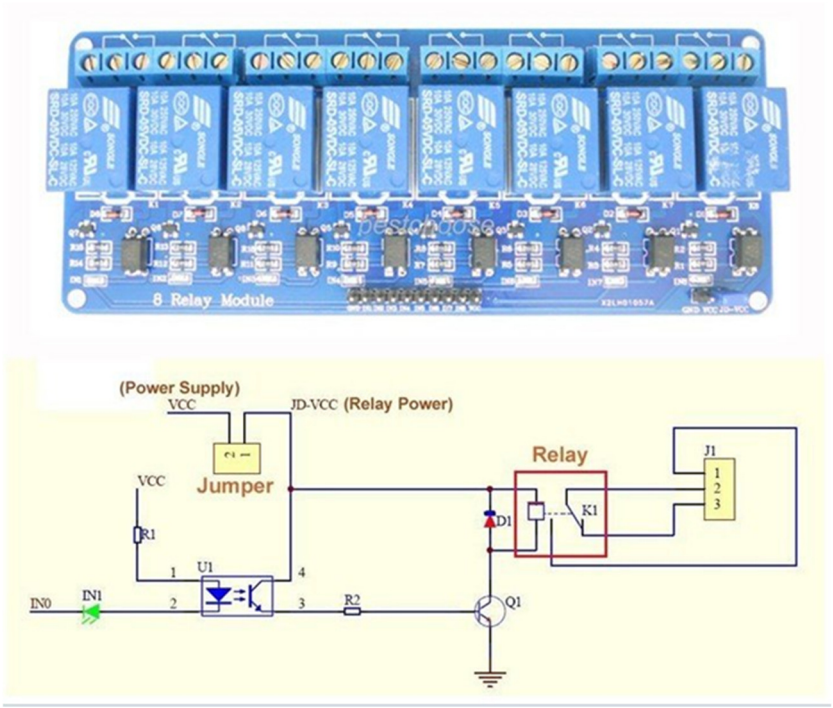

Figura 1.1_1: Bloque relés optoacoplados

La salida de las placas Arduino son perfectamente útiles para controlar cargas que no consuman demasiada corriente, como un Led, pero son insuficientes para cargas mayores. ¿Cómo hacemos para controlar por ejemplo una lámpara o un motor que se alimentan de 220 voltios? Una forma es emplear un módulo de relés como el que analizo en este artículo.

Antes de comenzar con el análisis: como sucede con muchos módulos para Arduino que se consiguen en el mercado, la placa que utilicé no tiene una marca que identifique a su fabricante ni información “oficial” sobre su funcionamiento. Seguramente provenga de alguna factoría china que produce estos productos que luego son vendidos por numerosas tiendas a través de la red y del cual pueden existir distintas variedades, con diferentes características. Antes de conectarlo según las indicaciones que se dan a continuación, por favor, asegúrense de que el módulo que tienen entre sus manos sea igual al que se describe aquí.

Hecha esta salvedad, comencemos con una descripción general de la placa. Se trata de un módulo de 4 relés (o relays) que funcionan a 5 Voltios, capaces de manejar cargas de hasta 10 Amperes en 250 Voltios, convenientemente aislados mediante optoacopladores de las entradas, las que cuentan con leds individuales que sirven como indicadores de estado.

Los distintos componentes del módulo pueden verse en la siguiente imagen:

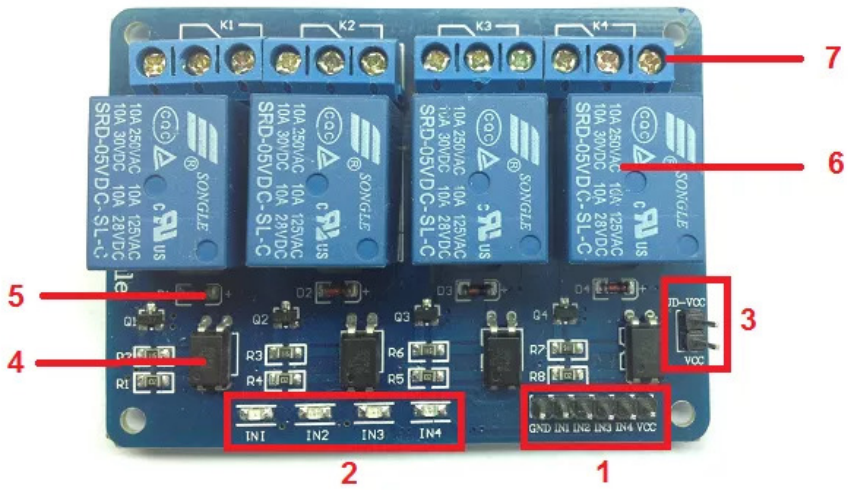

Figura 1.1_2: Partes del módulo

Como se puede apreciar, la placa tiene un conector de entradas (IN1 a IN4) y alimentación (GND es masa o negativo y Vcc es el positivo) [1], cuatro leds que indican el estado de la entradas [2], un jumper selector para la alimentación de los relés [3], cuatro optoacopladores del tipo FL817C [4], cuatro diodos de protección [5], cuatro relés marca SONGLE con bobinas de 5V y contactos capaces de controlar hasta 10 Amperes en una tensión de 250V [6] y cuatro borneras, con tres contactos cada una (Común, Normal abierto y Normal cerrado), para las salidas de los relés [7].

1.2.- Esquemático

En la figura 1.2_1 se puede apreciar el circuito esquemático de un canal, el resto de los canales repite la misma configuración.

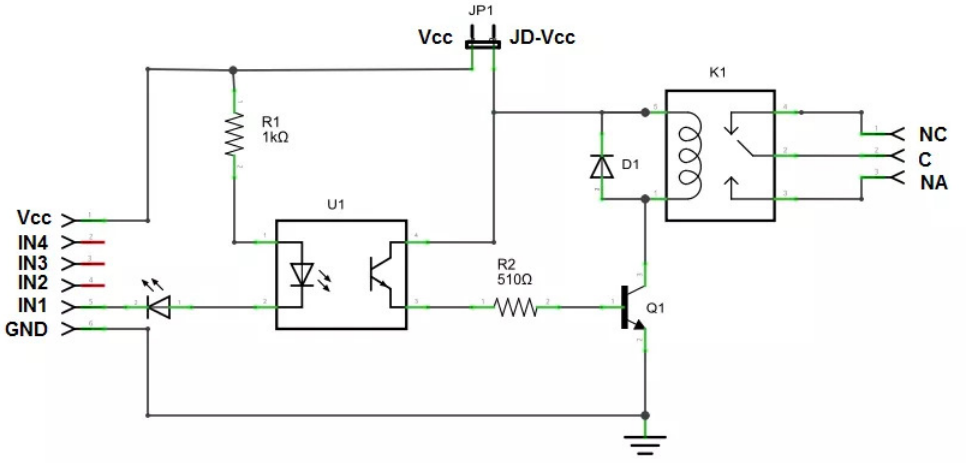

Figura 1.2_1: Esquema de un canal

1.3.- Funcionamiento

A partir del circuito de la Fig. 1.2_1 analicemos el funcionamiento del circuito: la entrada IN1 está conectada al cátodo del diodo del optoacoplador a través del led indicador. El ánodo del diodo del optoacoplador se conecta a Vcc (positivo) por intermedio de R1, una resistencia de 1000 ohms. Estos tres componentes, el diodo indicador, el diodo del opto y la R1 forman un circuito serie por el cual circula la corriente cuando la entrada está a un nivel BAJO (conectada a GND) y no circula si la entrada está a un nivel ALTO (conectada a Vcc).

El transistor del opto tiene su colector a JD-Vcc y su emisor conectado a Q1 a través de una resistencia de 510 ohms. Este es otro circuito serie por el cual circula corriente cuando el transistor del opto conduce al ser "iluminado" por su diodo, con lo que se introduce corriente en la base de Q1 a través de R2.

Finalmente, Q1 está conectado en una típica configuración emisor común, con su emisor a masa (GND) y la bobina del relé como carga en el colector. Cuando circula corriente por la base desde el opto, Q1 se satura permitiendo el paso de la corriente a través de la bobina del relé, lo que produce que se cierren los contactos del mismo (común con normal abierto). El diodo D1 protege al transistor de la tensión que aparece en la bobina del relé cuando deja de circular corriente por la misma.

En síntesis, al ponerse la entrada a nivel BAJO se pone a la saturación el transistor Q1 a través del optoacoplador con lo que se cierra el contacto normal abierto del relé.

1.4.- Alimentación y consumo

La forma mas sencilla de alimentar este módulo es desde Vcc y GND de la placa Arduino, manteniendo el Jumper en su lugar, con lo que JD-Vcc = Vcc. Esta conexión tiene dos limitaciones importantes:

- Se pierde la aislación eléctrica que brindan los optoacopladores, lo que aumenta la posibilidad de daño al Arduino si hay algún problema con las cargas de los relés.
- La corriente consumida por las bobinas de los relés debe ser provista por la placa Arduino. Cada bobina consume unos 90 mA y las cuatro juntas suman 360 mA. Si a esto le sumamos los consumos que pueden tener otras salidas, estamos muy cerca de los 500 mA que puede suministrar un puerto USB. En este caso se debería alimentar al Arduino con una fuente externa, lo que aumenta el limite de corriente a 1 A (en el caso de la Arduino UNO).

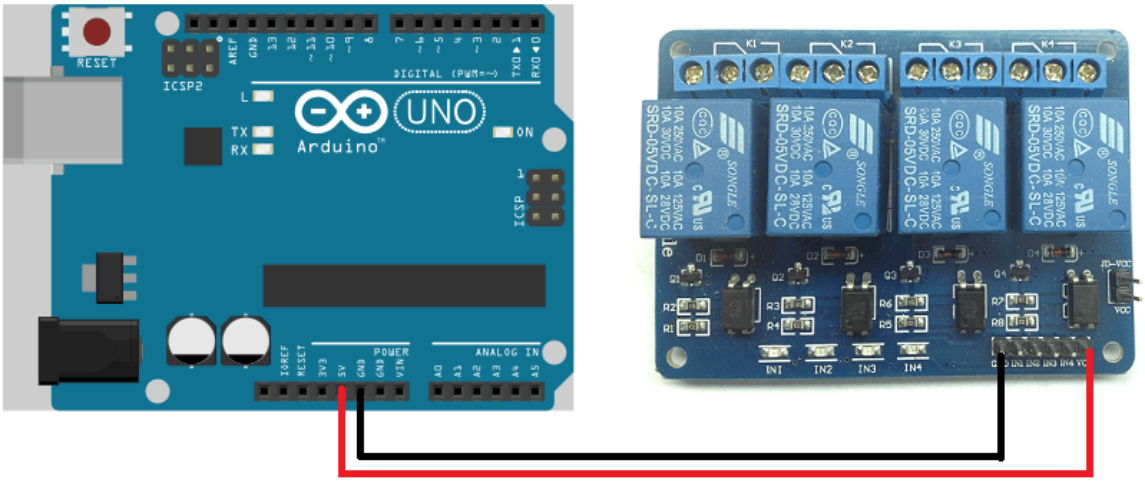

Figura 1.4_1: Alimentación relés

2.- ARDUINO DUE

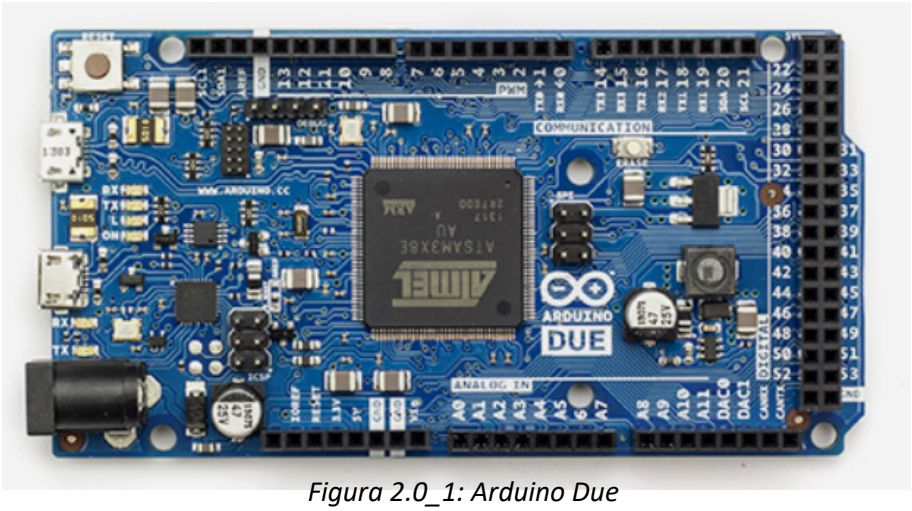

Figura 2.0_1: Arduino Due

Cuando se plantea el problema del cuadro de protecciones y control inteligente, se sabe que el microcontrolador elegido tiene que tener implementado un servidor que es el encargado de escuchar la entrada de clientes, en este caso el dispositivo Android, además tiene que ejecutar un muestreo de la señal a una velocidad con un periodo o tiempo de muestro de 1ms, tiene que procesar las mil muestras en el menor tiempo posible, y ser capaz de poder establecer una comunicación bidireccional entre cliente-servidor de 100 ms.

Para dicho acometido se piensa en un Cortex ARM M3 Figura 2.8_1, o un DSP de la familia C2000 de Texas Instruments.

El Arduino Due es la primera placa de desarrollo de Arduino basado en ARM Cortex M3, que posee un microcontrolador de 32 bits, programable mediante el familiar IDE de Arduino. La gran ventaja del entorno de Arduino, es la gran cantidad de librerías desarrolladas tanto para comunicaciones como en nuestro caso para crear los Sockets necesarios que permiten establecer la comunicación mediante el protocolo TCP/IP y UDP, así como otras muchas

como la librería del control de los Timers a bajo nivel que posee el Cortex M3 y otras muchas más como las comunicaciones serie I2c o SPI.

Arduino Due dispone de 54 pines digitales de entrada-salida (de los cuales 12 pueden utilizarse para salidas PWM), 12 entradas analógicas, 4 UARTs (Universal Asíncrona Recepción y transmisión Serie), un reloj de 84 MHz, una

conexión USB OTG, 2 DAC (Salidas digital a analógico), 2 TWI, un conector de alimentación, un cabezal SPI, un cabezal JTAG, un botón de reinicio y un botón de borrado. También hay algunas características interesantes como DACs, Audio, DMA, una biblioteca multitarea experimental y más.

Para poder compilar el código de este microcontrolador Cortex se ha utilizado la versión del IDE de Arduino 1.7.9. Todos los shields que implementen plenamente la disposición Arduino R3 son compatibles directamente (como el Arduino WiFi y Ethernet Shield).

Este microcontrolador junto con la plataforma Arduino se han elegido, por todo lo dicho anteriormente, plataforma software y librerías gratuitas, así como también las importantes prestaciones que ofrece el Cotex ARM M3 que puede ejecutar 84 millones de instrucciones simples por segundo, también la posibilidad importantísima de poder trabajar mediante Interrupciones y también la posibilidad de no solo poder visualizar resultados en Simulink sino también, la posibilidad de utilizar la modalidad deploy to hardware, que permite hacer pruebas de programación modular y cargarla directamente al microcontrolador .

Características:

- Microcontrolador: AT91SAM3X8E
- Voltaje de operación: 3.3V
- Voltaje de entrada recomendado: 7-12V
- Voltaje de entrada min/max: 6-20V
- Digital I/O Pins: 54 (de los cuales 12 proveen salida PWM)
- Analog Input Pins: 12
- Analog Outputs Pins: 2
- Corriente total de salida DC en todas las lineas I/O: 130 mA
- Corriente DC para el Pin de 3.3V: 800 mA
- Memoria Flash: 512 KB disponibles para aplicaciones del usuario
- SRAM: 96 KB (two banks: 64KB and 32KB)
- Clock Speed: 84 MHz

Esta información técnica se ha obtenido de la página web: http://arduino.cl/arduino-due/

3.- TARJETA ETHERNET ARDUINO

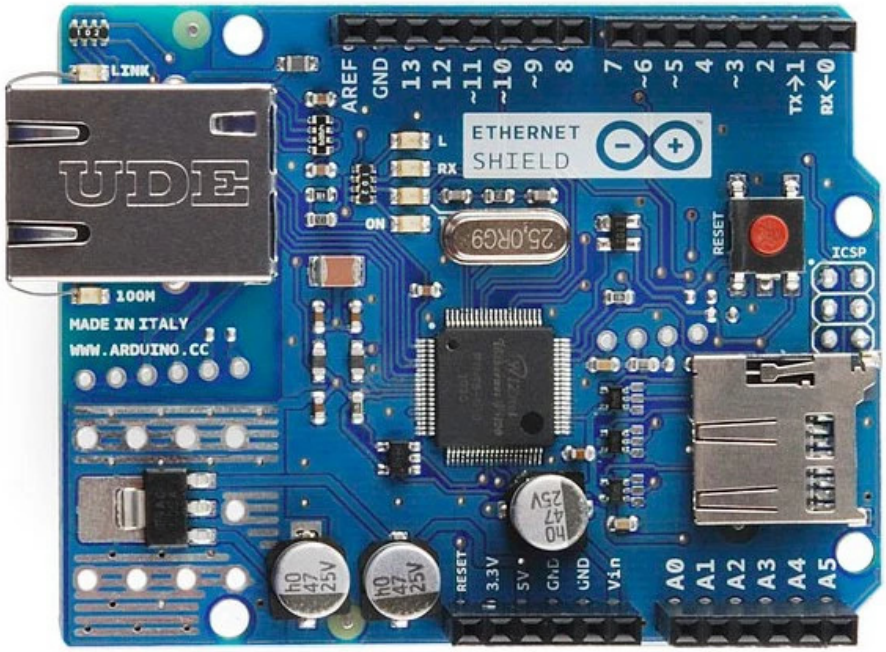

Figura 3.0_1: Shield Ethernet Arduimo

El Arduino ethernet shield nos da la capacidad de conectar un Arduino a una red ethernet. Es la parte física que implementa la pila de protocolos TCP/IP.

Está basada en el chip ethernet Wiznet W5100. El Wiznet W5100 provee de una pila de red IP capaz de soportar TCP y UDP. Soporta hasta cuatro conexiones de sockets simultáneas. Usa la librería Ethernet para leer y escribir los flujos de datos que pasan por el puerto ethernet. Me permitirá escribir sketches que se conecten a internet usando la shield.

Una característica importante de este módulo es que la comunicación con Arduino Due se hace a través de la cabecera ICSP que utiliza el bus de comunicaciones SPI, dejando libre la parte superior del módulo wifi para poder apilar en este caso el driver que controla los Motores.

4.- ROUTER TP-LINK TL-MR3020

Figura 4.0_1: Shield Ethernet Arduimo

- Comparta su conexión 3G/4G*, compatible con múltiples módems USB 3G/4G UMTS/HSPA+/HSPA/EVDO probados en condiciones reales.
- El tamaño y peso del TL-MR3020 lo convierten en el equipo ideal para llevárselo de viaje y compartir su conexión móvil en cualquier lugar con cobertura 3G/4G*.
- Velocidad inalámbrica hasta 150 Mbps.
- Funciona de tres formas diferentes según la situación: como router 3G/4G*, router cliente WISP y router de viaje.(AP Modelo).
- El TL-MR3020 le permite estar conectado a Internet permanentemente gracias a su modo de conexión redundante 3G/4G y WAN.

5.- APARAMENTA EQUIPO AXILIAR DEL CONTACTOR

Product data sheet
Characteristics

A9C15924
Acti 9 iACT - 24V DC control and auxiliary contact
1 NO with Ti24 PLC interface

Main

Range	Acti 9
Product or component type	Control and indication auxiliary
Device short name	IACT24
Signal contacts composition	1 NO
Signalling circuit voltage	24 V DC : - 20...20 % 0.002...0.1 A
[Uc] control circuit voltage	24 V DC falling edge 24 V DC rising edge 230 V AC 50/60 Hz maintained >= 200 ms
Control type	Remote control
[Us] rated supply voltage	230 V AC 50/60 Hz

Complementary

Mounting mode	Fixed, on the left of CT
Mounting support	DIN rail
Comb busbar and distribution block compatibility	Bottom : YES Top : NO
9 mm pitches	2
Height	84 mm
Width	18 mm
Depth	60 mm
Colour	White
Immunity to microbreaks	10 ms
Undervoltage behaviour	No opening if loss of 24 V DC
Electrical durability	250000 cycles conforming to IEC 60947-5-1
Connections - terminals	Control circuit : tunnel type terminals for 2 flexible cable(s) 2.5 mm² Control circuit : tunnel type terminals for 2 rigid cable(s) 1.5 mm² Auxiliary supply : tunnel type terminals for 2 flexible cable(s) 2.5 mm² Auxiliary supply : tunnel type terminals for 2 rigid cable(s) 1.5 mm² Control circuit : spring-loaded terminal Ti24 for 2 flexible cable(s) 0.25...0.75 mm² Control circuit : spring-loaded terminal Ti24 for 1 flexible cable(s) 0.5...1.5 mm²

Aug 10, 2017

6.- APARAMENTA CONTACTOR

Hoja de datos del producto
Características

A9C21732
Contactor modular iCT 25A 2NO 230...240V 50Hz MO

Principal

Gama	Acti 9
Nombre del producto	Acti 9 iCT
Tipo de producto o componente	Conector
Nombre corto del dispositivo	ICT
Aplicación de dispositivo	Motor-heating-lighting
Número de polos	2P
Composición de polos de contacto	2 NO
Tipo de red	CA
Categoría de utilización	AC-1 de acuerdo con EN 60947-4-1 AC-3 de acuerdo con EN 60947-4-1 AC-7A de acuerdo con EN 61095 AC-7A de acuerdo con IEC 1095 AC-7B de acuerdo con EN 61095 AC-7B de acuerdo con IEC 1095
Tipo de control	Control manual y remoto
Tensión del circuito de control	230...240 V CA 50 Hz

Complementario

[Ie] intensidad de funcionamiento nominal	25 A AC-7A 8.5 A AC-7B
Frecuencia de red	50/60 Hz
[Ue] tensión de funcionamiento nominal	250 V CA 50 Hz
Potencia máxima	1.2 W 250 V AC
[Ui] tensión nominal de aislamiento	500 V CA 50/60 Hz
[Uimp] tensión nominal soportada al impulso	4 kV
Tipo de señal de control	Mantenido
Frecuencia de conmutación	100 maniobras de conmutación/día
Señalizaciones en local	Indicador de acción
Consumo de energía en espera VA	2.7 VA
Potencia de entrada en VA	9.2 VA
Modo de montaje	Ajustable en clip
Soporte de montaje	Carril DIN simétrico de 35 mm
Pasos de 9 mm	2
Altura	81 mm
Anchura	18 mm
Profundidad	60 mm
Color	Blanco
Durabilidad eléctrica	200000 ciclos CA 50/60 Hz de acuerdo con EN 61095 200000 ciclos CA 50/60 Hz de acuerdo con IEC 1095
Conexiones - terminales	Circuito de control : 2 terminales de tipo túnel 1,5 mm² para rígido cable(s) Circuito de alimentación : 1 terminales de tipo túnel 1...4 mm² para Flexible cable(s) Circuito de alimentación : 1 terminales de tipo túnel 1.5...6 mm² para rígido cable(s) Circuito de control : 1 terminales de tipo túnel 1.5...2.5 mm² para rígido cable(s) Circuito de control : 2 terminales de tipo túnel 1.5...2.5 mm² para Flexible cable(s)

7.- APARAMENTA EQUIPO AUXILIAR DE SEÑALIZACIÓN

Product data sheet
Characteristics

A9A26897
iOF+SD24 open/close & fault indicating auxiliary contact with Ti24 top connector

Main

Range	Acti 9
Product or component type	OC and fault contact with PLC interface
Device short name	IOF+SD 24
Signal contacts composition	1 NO + 1 NC
[Ie] rated operational current	2...100 mA at 24 V DC ± 20%
9 mm pitches	1

Complementary

Range compatibility	Acti 9 iC60 Acti 9 iDPN Vigi Acti 9 iID Acti 9 iSW-NA Acti 9 iC60 RCBO Acti 9 iC65 Acti 9 ARA Acti 9 RCA Acti 9 iDPN (China version) Connectors Ti24
[Ui] rated insulation voltage	500 V conforming to IEC 60664-1
[Uimp] rated impulse withstand voltage	6 kV conforming to IEC 60664-1
Local signalling	Mechanical indicator
Mounting mode	Clip-on
Mounting support	DIN rail
Height	88 mm
Width	9 mm
Depth	73.5 mm
Product weight	25 g
Colour	White
Electrical durability	20000 cycles conforming to IEC 60947-5-4
Connections - terminals	Spring-loaded terminal Ti24 in top 2 cable(s) 0.5...1.5 mm² flexible

Aug 19, 2017

8.- APARAMENTA CONTROL A DISTANCIA DEL DIFERENCIAL

Hoja de datos del producto
Características

A9C70122
Mando motorizado RCA para iC60 con interfaz Ti24 - 1P, 1P+N, 2P

Principal

Gama	Acti 9
Tipo de producto o componente	Auxiliar de control remoto
Nombre corto del dispositivo	RCA iC60
Composición de contactos de señalización	1 OF contactos sin tensión 1 OF Ti24 1 SD Ti24
Tensión del circuito de señalización	Contacto sin tensión 24...230 V 0.01...1 A CA Ti24 24 V 100 mA CC

Complementario

Tipo de control	Mando a distancia impulso y sostenido 230 V CA 50/60 Hz Mando a distancia mantenido Ti24 24 V CC
Corriente de entrada	5 mA impulso 5 mA mantenido 5.5 mA muy baja tensión sostenida
Duración del impulso	>= 200 ms circuito de control
[Us] tensión de alimentación nominal	230 V AC 50/60 Hz
[Ui] tensión asignada de aislamiento	400 V
[Uimp] tensión nominal soportada al impulso	6 kV
Valor de ajuste	Auto : control remoto autorizado Apagado : control remoto inhibido Modo 1 : central control Modo 3 : control central y anulación local Posición A : reconexión autorizada tras disparo Possición B : reconexión inhibida tras disparo
Señalizaciones en local	Parpadeo naranja LED : no está en funcionamiento Parpadeo verde LED : listo para operar
Modo de montaje	Fijo
Soporte de montaje	Carril DIN
Compatibilidad de bloque de distribución de embarrado tipo peine	NO
Pasos de 9 mm	7
Altura	93 mm
Anchura	63 mm
Profundidad	75,5 mm
Peso del producto	0.4 kg
Color	Blanco
Durabilidad eléctrica	10000 ciclos
Endurancia mecánica	10000 ciclos
Preparado para candado	Con candado
Longitud de cable pelado para conectar bornas	Alimentación : 10 mm (inferior) Entrada : 10 mm (inferior) Salida : 8 mm (superior) Ti24 interface : 10 mm (terminal)
Par de apriete	Salida auxiliar : 0.7 N.m

9.- APARAMENTA BLOQUE DIFERENCIAL

Hoja de características del producto
Características

A9Q11225
Bloque diferencial Vigi iC60 - 2P - 25A - 30mA - clase AC

Principal

Gama	Acti 9
Tipo de producto o componente	Add-on residual current devices
Nombre corto del dispositivo	Vigi iC60
Número de polos	2P
Intensidad nominal (In)	25 A (-5...60 °C)
Sensibilidad de fuga a tierra	30 mA
Retardo de la protección contra fugas a tierra	Instantáneo
Clase de protección contra fugas a tierra	Type AC
Tipo de red	CA
Frecuencia de red	50/60 Hz
[Ue] tensión de funcionamiento nominal	230/400 V CA 50/60 Hz coordinación EN 61009-1 230/415 V CA 50/60 Hz coordinación IEC 61009-1
Normas	EN 61009-1 IEC 61009-1
Pasos de 9 mm	3

Complementario

Compatibilidad de la gama	Acti 9 iC60 Acti 9 Reflex iC60
Ubicación del dispositivo en el sistema	Salida
Protección contra sobretensión	Sin
Tecnologia de disparo corriente residual	Independiente de la tensión
[Ui] tensión nominal de aislamiento	500 V CA 50/60 Hz coordinación IEC 60947-2
[Uimp] tensión nominal soportada al impulso	6 kV coordinación IEC 60947-2
Señalizaciones en local	Indicador de disparo

10.- APARAMENTA INTERRUPTOR AUTOMÁTICO

Hoja de datos del producto
Características

A9C62210
Interruptor automático con telemando Reflex iC60N - con Ti24 - 10 A 2P curva C

Principal

Aplicación de dispositivo	Control Distribución
Gama	Acti 9
Nombre del producto	Reflex iC60
Nombre corto del dispositivo	STD1000RL-SAE
Tipo de producto o componente	Interruptor automático de control integrado
Número de polos	2P
Número de polos protegidos	2
Tipo de red	CA
Tecnología de unidad de disparo	Térmico-magnético
Código de curva	C

Complementario

Intensidad nominal (In)	10 A
Frecuencia de red	50/60 Hz
Código de poder de corte	N
Poder de corte	Icu 20 kA en 220...240 V CA 50/60 Hz de acuerdo con EN/IEC 60947-2 Icu 10 kA en 380...415 V CA 50/60 Hz de acuerdo con EN/IEC 60947-2
[Ui] tensión nominal de aislamiento	500 V CA
Indicador de posición del contacto	Sí
Tipo de control	Control remoto por mando continuo Botón ON/OFF Control remoto por impulso Maneta
Tipo de señal de control	Mantenido Impulso Muy baja tensión sostenida
Señalizaciones en local	LED en rojo: fallo LED intermitente: disparo del dispositivo LED verde: dispositivo ON LED intermitente: dispositivo listo LED intermitente: recalentamiento del dispositivo
Tensión del circuito de control	230 V CA 50/60 Hz 24 V CA 50/60 Hz 24 V DC 48 V CA 50/60 Hz 48 V DC
[Us] tensión de alimentación nominal	230 V AC 50/60 Hz
Modo de montaje	Fijo
Soporte de montaje	Carril DIN simétrico de 35 mm
Pasos de 9 mm	9
Altura	84 mm
Anchura	81 mm
Profundidad	77 mm
Color	Blanco
Endurancia mecánica	50000 ciclos
Durabilidad eléctrica	AC-1 : 50000 ciclos CA 50 Hz
Preparado para candado	Con candado
Descripción de las opciones de bloqueo	Candado en posición OFF

11.- DIAGRAMAS DE FLUJO O FLUJOGRAMAS

11.1.- Introducción Concepto

Los diagramas de flujo son una manera de representar visualmente el flujo de datos a través de sistemas de tratamiento de información. Los diagramas de flujo describen que operaciones se realizan en un programa, la secuencia de las mismas y las decisiones lógicas tomadas para solucionar un problema dado.

Un diagrama de flujo u organigrama es una representación diagramática que ilustra la secuencia de las operaciones que se realizarán para conseguir la solución de un problema. Los diagramas de flujo se dibujan generalmente antes de comenzar a programar el código frente a la computadora. Los diagramas de flujo facilitan la comunicación entre los programadores y la gente del negocio. Estos diagramas de flujo desempeñan un papel vital en la programación de un problema y facilitan la comprensión de problemas **complicados** y sobre todo **muy largos**. Una vez que se dibuja el diagrama de flujo, llega a ser fácil escribír el programa en cualquier idioma de alto nivel. Vemos a menudo cómo los diagramas de flujo nos dan ventaja al momento de explicar el programa a otros. Por lo tanto, está correcto decir que un diagrama de flujo es una necesidad para la documentación mejor de un programa complejo.

11.2.- Las Reglas para la creación de Flujogramas

- Los Diagramas de flujo deben escribirse de arriba hacia abajo, y/o de izquierda a derecha.
- Los símbolos se unen con líneas, las cuales tienen en la punta una flecha que indica la dirección en la que fluye la información del procesos, se deben de utilizar solamente líneas de flujo horizontal o verticales (nunca diagonales).
- Se debe evitar el cruce de líneas.
- No deben quedar líneas de flujo sin conectar
- Todo texto escrito dentro de un símbolo debe ser legible, preciso, evitando el uso de muchas palabras.
- Todos los símbolos pueden tener más de una línea de entrada, a excepción del símbolo final.
- Solo los símbolos de decisión pueden y deben tener más de una línea de flujo de salida.

Los símbolos más utilizados en los flujogramas son:

– Terminal (**rectángulo curvilíneo**). Empleado al principio y al final de las acciones del flujograma. Delimita cada bloque de programación.

– Operación **(rectángulo**). Indica la realización de una operación/acción determinada. Por ejemplo, sumar, inicializar unos registros a valor determinado, etc.

– Línea de Flujo **(línea con flecha).** Indica el camino operativo de las uniones entre las operaciones/acciones del flujograma.

– Toma de Decisión **(rombo)**. Permite la bifurcación de la secuencia ordenada de operaciones cuando se cumple una condición determinada (saltos condicionales).

– Etiqueta (**círculo**). Permite el enlace entre Líneas de Flujo de un flujograma que ocupa más de una página. Dentro del circulo debe contener un nombre, que identifica a la etiqueta y en otra página debe aparecer una etiqueta con el mismo nombre, esto indica el enlace entre dichas Líneas de Flujo en diferentes páginas.

– Rutina/Función (**rectángulo de lados dobles**). Permite la identificación de la rutina que es ejecutada en ese punto del programa. Dentro se escribe el nombre de la rutina, el cual debe coincidir con el que contenga el símbolo.

Algo muy interesante cuando se trabaja paralelamente con el flujograma y el código del lenguaje de programación que se esté utilizando, es que en el proceso de depuración se recurre al flujograma y al código de forma iterativa. Esto implica la posibilidad de darse cuenta de los errores producidos en el código e incluso la elaboración de distintas estrategias que suelen dar como resultado final un código más eficiente.

http://www.mis-algoritmos.com/aprenda-a-crear-diagramas-de-flujo
Flujogramas: Diseño Electrónico Avanzado [Master en Ingeniería Mecatrónica]

12.- PROGRAMACIÓN EN ANDROID
12.1.- Introducción Conceptos

Esta introducción no pretende ser un curso de Android, si no que pretende comentar las ideas básicas de la programación en Android, para poder entender la aplicación cliente Android de este proyecto, los métodos utilizados y el concepto de hilos de programación que ha sido necesario poner en práctica.

Podemos resumir que en programación Android, toda aplicación está constituida por tres partes claramente diferenciadas, por una parte está el ***Layout.xml*** (vista gráfica), por otro lado la ***Activity(.java)*** (métodos-código lógico) y finalmente un fichero denominado ***AndroidManifest.xml***, que también es tipo Layout.

La finalidad del fichero ***AndroidManifest.xml*** es declarar una serie de metadatos de la aplicación que el dispositivo debe conocer antes de instalar la aplicación. En él se indican el nombre del paquete en el que por defecto se buscarán las actividades que declaremos en nuestra aplicación. El nombre de la aplicación, las actividades que conforman la aplicación y cuál es la principal, el ícono de nuestra aplicación entre otros, y sobre todo los permisos que serán concedidos de manera explícita por el usuario, si da su consentimiento para instalar la aplicación.

Al crear un nuevo proyecto por una parte creamos una *Activity (*Actividad).java, que es la que contendrá todos los métodos de la aplicación principal, es decir toda la parte lógica de la programación y donde están contenidos los distintos hilos de programa. El único método que se sobrescribe (que se crea por defecto) de la clase padre es **onCreate**.

Y lo primero que se hace es llamar al **onCreate** de la clase *Activity.* Esto se hace así porque cuando una actividad se crea, se puede querer crear con información que tenía previamente. Esta información se recibe en el objeto **savedInstanceState**. Imaginemos por ejemplo que cambiamos la posición del móvil a apaisado mientras rellenamos un formulario. Este cambio provoca que la actividad se creé de nuevo, y lo más usable, es que los campos que ya haya rellenado el usuario sigan rellenos con la misma información, y para esto, es necesario este método.

A continuación se realiza algo importante que es vincular la parte digamos del código *Activity*.java con los objetos gráficos que contendrá el Layout.xml(vista gráfica) de nuestra aplicación. Esto se realiza mediante el método **setContentView**. Este método solo se puede llamar desde **onCreate**, y además es un campo immutable. Solo se puede fijar la vista de la actividad una sola vez. Es la forma que tiene **Android** de obligarnos a tener una única vista por actividad.

Acontinuación se muestra el código explicado:

```
package com.autentia.android;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;

public class FirstActivity extends Activity {

    @Override
    public void onCreate(Bundle savedInstanceState) {
           super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
    }
}
```

Por otro lado tenemos la vista (layout.xml) que es donde se implementa toda la parte gráfica botones, ventanas, sliders etc. Como se comentó anteriormente la *Activity* y el *layout* quedan vinculados mediante el método **setContentView,** que llamamos desde **onCreate.**

A partir de aquí empieza el ciclo de vida de la *Activity.java.* El sistema llama al constructor de la actividad mientras que también inicia la aplicación si es necesario y llama a los siguientes métodos por orden:

- onCreate
- onStart
- onResume

Hasta este momento lo que ha ocurrido es que hemos dada vida a la actividad. Pero en cualquier momento podemos minimizar, dejar la aplicación en segundo plano porque queremos ejecutar otra actividad, y entonces empieza la segunda parte del ciclo de vida de una actividad. Esto ocurre cuando por ejemplo el usuario pulsa el botón de retorno desde la actividad. Los métodos que son llamados y por el siguiente orden son:

- onPause
- onStop
- onDestroy

Ejemplos de acciones que destruirán una actividad:

- Cambio del móvil a apaisado o viceversa
- La actividad ya no se ve en la pantalla, o el sistema está bajo de recursos
- El usuario presiona el botón de retroceso o de Home y sale de la aplicación

Después de esto la aplicación es cerrada.

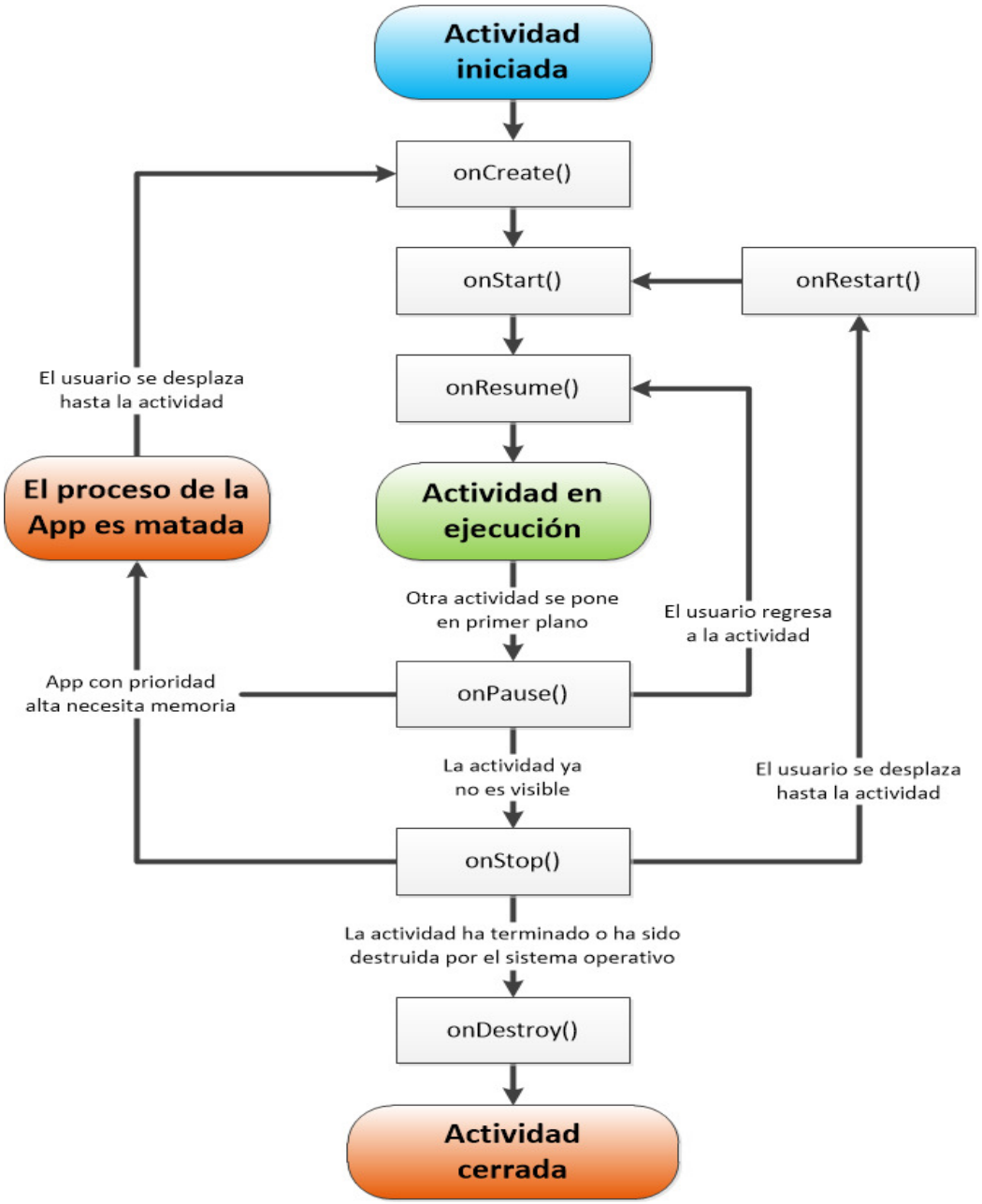

Figura 12.1_1: Ciclo de vida de una Actividad oficial de Android

A continuación se pasa a describir cada uno de estos métodos del ciclo de vida para posteriormente comentar y justificar los métodos especiales utilizados en la aplicación Cliente Android de este proyecto.

onStart:

Se llama immediatamente después del onCreate. Si nuestra aplicación estaba en segundo plano, **onStart** será llamado cuando la aplicación vuelva a estar en primer plano.

onResume:

Es el último método que se llama antes de que la actividad tenga acceso a la pantalla. Si algún elemento del interfaz gráfico ha cambiado mientras la actividad estaba en segundo plano este método es el sitio para asegurar que el estado está sincronizado. No importa de que estado venga, cuando la actividad vuelva a estar en primer plano este método será llamado.

onPause

Es el primer método que se llama cuando la aplicación se está yendo de la pantalla. Si tenemos bucles, procesos, animaciones que deberían estar corriendo cuando la actividad está en pantalla este método es el idóneo para pararlos. Este método también se llama cuando lanzamos otra actividad desde la que se está ejecutando actualmente. Este método es importante porque puede ser el único en avisarnos de que la actividad o incluso toda la aplicación se está cerrando. En este método deberíamos guardar cualquier información importante a disco, base de datos o preferencias**.**

onStop

Cuando se llama a onStop lo que sabemos es que la actividad está **oficialmente fuera de pantalla**. No siginifica que la actividad se esté apagando, aunque podría ser. Solo se puede asumir que el usuario ha dejado tu actividad por otra. Si estás haciendo algún proceso que solo debería estar corriendo cuando la actividad está en ejecución este es un buen momento para pararla.

onDestroy

Es el último método que se llama antes del final. Es la última oportunidad para limpiar lo necesario antes de que el propio sistema la elimine por completo. Cualquier proceso de *background* que la actividad puede taner corriendo debe pararse. Sin embargo porque este método se haya llamado no significa que la actividad sea borrada. Si tienes algún hilo corriendo, este puede seguir corriendo y consumiendo recursos incluso aunque este método se llame.

12.2.- Metodos especiales utilizados en la Aplicación Cliente Android

runOnUiThread ():

Este método arranca lo que se conoce un hilo secundario de programa cuando se la llama desde el hilo principal de programa, que permite no bloquear a la aplicación principal, pues de ello se encarga el planificador de tareas del sistema operativo. Fundamentalmente este método está pensado para actuar sobre una vista, es decir para cambiar o actualizar un valor de una etiqueta de texto, o como en nuestro caso, ha servido para cada 2 segundos comprobar si hay red, para posteriormente actuar sobre la interfaz gráfica cambiando si hay red, el semáforo Red en verde, y en caso contrario en rojo.

Existe otros métodos similares como **Handler.post(Runnable r)**, que sirven para lo mismo, pero en este caso con la diferencia de que con este método, podemos acceder a los componentes o variables de otros *tread* y no solo al del hilo principal.

Para que un hilo secundario no detenga el hilo principal de programa, el camino correcto a seguir es renderizar la interfaz de la aplicación y al mismo tiempo ejecutar en segundo plano la otra actividad para continuar con la armonía de la aplicación y evitar paradas inesperadas.

Es aquí donde entran en juego los hilos, porque son los únicos que tienen la habilidad especial de permitir al programador generar concurrencia en sus aplicaciones y la sensación de multitareas ante el usuario. Por eso la programación en Android está muy ligada al concepto de multi-hilo, y existen muchos métodos que permiten ejecutar hilos con propósitos diferentes que se adaptan a las exigencias de la tarea demandad con al hilo.

A continuación se ilustra esta idea con la Figura 5.2_1:

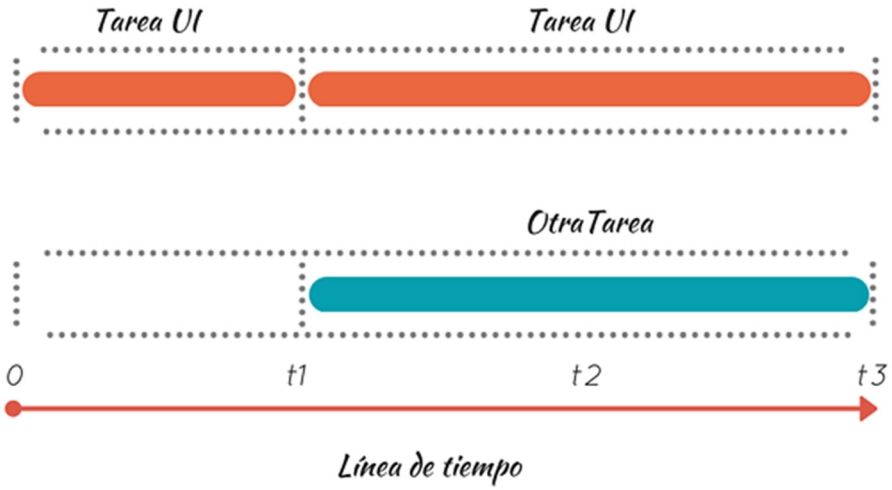

Figura 12.1_2: Hilos y Concurrencia

AsyncTask :

Función o metodo en nuestro programa:

```
private class MySocket extends AsyncTask<String, Integer, int []>
```

Cuando existe la posibilidad de lanzar una tarea bloqueante y largar, como es el caso de lanzar el socket que permite establecer la comunicación en este proyecto, con el Robot, es inprescidible implementarlo de esta forma.

En los casos en los que se ejecutan varias instrucciones que deben presentar cambios en el hilo principal. Si se aplica el enfoque anterior utilizando los metodos **runOnUiThread ():,** por un lado para actualizar la interfaz gráfica, y por otro para llamar al socket, recoger la información y luego actualizarla mediante metodos del tipo **Handler.post(Runnable r)** .

El código para el envío de las ejecuciones tiende a ser muy largo, confuso y poco maleable a la hora de mantenimiento.

Por esta razón ha sido creada la interfaz AsyncTask, cuyo objetivo es liberar al programador del uso de hilos, la sincronización entre ellos y la presentación de resultados en el hilo primario. Esta clase unifica los aspectos relacionados que se realizarán en segundo plano y además gestiona de forma asíncrona la ejecución de las tarea.

Los métodos que conforma la **AsyncTask** son:

onPreExecute(): En este método van todas aquellas instrucciones que se ejecutarán antes de iniciar la tarea en segundo plano. Normalmente es la inicialización de variables, objetos y la preparación de componentes de la interfaz.

doInBackground(Parámetros…): Recibe los parámetros de entrada para ejecutar las instrucciones específicas que irán en segundo plano, luego de que haya terminado onPreExecute(). Dentro de él podemos invocar un método auxiliar llamado publishProgress(), el cual transmitirá unidades de progreso al hilo principal. Estas unidades miden cuanto tiempo falta para terminar la tarea, de acuerdo a la velocidad y prioridad que se está ejecutando.

En nuestro caso se ha utilizado para llamar al método que *socket,* que es el que establece la comunicación con el robot y empieza el flujo bidireccional de información entre el cliente(Android), y el servidor (Robot).

onProgressUpdate(Progreso…): Este método se ejecuta en el hilo de UI luego de que publishProgress() ha sido llamado. Su ejecución se prolongará lo necesario hasta que la tarea en segundo plano haya sido terminada. Recibe las unidades de progreso, así que podemos usar algún View para mostrarlas al usuario para que este sea consciente de la cantidad de tiempo que debe esperar.

onPostExecute(Resultados…): Aquí puedes publicar todos los resultados retornados por doInBackground() hacia el hilo principal. En nuestro caso sirve para actualizar las variables que y el vector que contiene tanto las coordenadas xc como yc, que posteriormente permiten graficar la información en la interfaz gráfica.

https://developer.android.com/reference/android/app/

http://jarroba.com/activity-entender-y-usar-una-actividad/

https://www.adictosaltrabajo.com/tutoriales/inicio-android/#01