Programador de Aplicaciones Informáticas

Proyecto: Mindstorms NXT 2.0

Alumnos:

Fátima Barros Barros

Adrio González González

Roberto Priegue Lago

ÍNDICE

1. Lego Mindstorm NXT 2.0

El Lego Mindstorm NXT 2.0 es la generación "NXT" de construcción de robots programables.

LEGO MINDSTORMS NXT está de vuelta, con modelos de robots nuevos, programación aún más personalizable, y con toda una nueva tecnología, entre la que se incluye un sensor de color.

LEGO Mindstorms NXT 2.0 combina la ilimitada versatilidad del sistema de construcción de LEGO con un ladrillo de microcomputadoras inteligente.

Entre los componentes del kit de montaje se distingue que el NXT de LEGO cuenta con un ladrillo microprocesador de 32 bits, una pantalla de matriz grande. Además dispone de 4 entradas y 3 puertos de salida, y comunicación por Bluetooth y conexión USB.

También contiene 3 servo-motores interactivos, cuatro sensores, entre los que se encuentran un sensor ultrasónico, dos sensores de contacto y el sensor de color totalmente nuevo.

Éste último presenta una triple funcionalidad: distingue los colores (negro, blanco, rojo, verde, azul y amarillo), la configuración de la luz, y funciona como una lámpara(es posible controlar los LED's del sensor para que ilumine con verde, rojo, amarillo,…).

Dispone de su propio software (PC y Mac) fácil de usar, basados en la programación por iconos de arrastrar y soltar, con 16 modelos de construcción diferentes y con distintos retos de programación.

El kit se presenta con 612 piezas que permiten innumerables montajes, y un CD con el software y los drivers necesarios.

Lo más innovador del nuevo software es que tiene editor de sonido, este graba cualquier tipo de sonido y después se programa para que el Ladrillo NXT pueda decir el sonido anteriormente grabado.

También contiene un editor de imagen, con el que se puede subir una imagen para que el Ladrillo NXT lo represente en su pantalla.

Mediante el uso de la conexión por medio de Bluetooth se puede obtener mando directo sobre el robot desde la computadora. Con lo que se podría crear una aplicación de control remoto.

Se puede utilizar en modo de sensor de luz: En este modo, actúa como el sensor de luz antigua: devuelve un valor de 0-100 donde 0 es más oscuro, y el 100 es más brillante. En este modo, también es posible elegir un LED de color deseado (de nuevo, rojo, azul o verde).

Incluye 2 Sensores de tacto, lo que le da al robot sentido del tacto; incluye 1 Sensor Ultrasónico, que podría ser resumido como los "ojos" del robot; incluye 3 Motores que dotas de movimiento al robot; y también incluye el ladrillo NXT, que vendría a ser el cerebro del robot.

1.1. Sensores

Los sensores son los dispositivos que se añaden a la estructura y que permite al robot "visualizar" la realidad que le rodea. De esta manera el robot será capaz de detectar un cambio en el color de una superficie, detectar un objeto, detectar que ha alcanzado el límite de una mesa…

El kit de lego Mindstrom NXT 2.0, adquirido para la realización del proyecto, contiene 4 sensores de tres tipos diferentes.

1.1.1. Sensor de color

El sensor de color será uno de los encargados de darle visión al robot (El otro será el sensor de ultrasonidos). En realidad, el sensor de color tiene tres funciones en una. Por un lado, el detector permite distinguir color, así como la intensidad de la luz (claro/oscuro). El detector es capaz de detectar 6 colores diferentes, leer la intensidad de luz en una habitación y medir la intensidad de luz de las superficies coloreadas. Además, el sensor de puede ser empleado a modo de lámpara, proyectando diferentes colores.

Una de las utilidades que se le puede dar a este sensor, y que será la empleada en el proyecto, será la de seguir una línea de un determinado color. Otro uso posible sería el de reaccionar de una determinada manera dependiendo del color que se detecte. La otra manera en la que se podría emplear este sensor sería la de indicador luminoso, dándole así un punto extra de personalidad.

Para que el funcionamiento del detecto, a la hora de detectar colores, sea óptimo, el sensor debería estar sostenido en el ángulo correcto a aproximadamente 1 cm de la superficie. Las lecturas incorrectas de colores pueden darse si el sensor se sujeta en otros ángulos, respecto de la superficie, o si es usado en un ambiente muy luminoso.

1.1.2. Sensor de ultrasonidos

El sensor de ultrasonidos es el otro sensor capaz de proporcionar el sentido de la vista al robot. Este sensor permite al robot ver y detectar objetos. También puede usarse para crear un robot capaz de sortear objetos, medir distancias y detectar movimientos.

Internamente, este sensor está formado por dos transductores de ultrasonidos, un emisor y un receptor. Se trata de un sensor complejo que requiere de su propio microprocesador. El sensor trabaja como un sonar, enviando un pulso de ultrasonido de 40kHz y midiendo el tiempo que tarda el sonido en viajar hacia un objeto, reflejarse y volver.

El sensor de ultrasonidos mide distancias en centímetros y en pulgadas. Permite medir distancias de entre 0 y 255 cm con una precisión de +/- 3 cm calculando el tiempo que tarda una onda sonora en chocar con un objeto y volver, como un eco. Dependiendo de la forma y el material de que este hecho el objeto, será más fácil o no detectar el objeto. Por ejemplo, si el objeto es muy grande y de superficie dura devolverá una lectura muy buena, mientras que un objeto curvado o muy fino será más difícil de detectar por el sensor.

1.1.3. Sensor de contacto

El sensor de contacto fabricado por Mindstorms es otro de los cuatro sensores básicos que vienen incluidos en el pack cuando se adquiere el NXT. Posiblemente se trate del sensor más sencillo de todos. El sensor es básicamente en un interruptor que nos devuelve un valor de 1 mientras está pulsado, o un valor de 0 mientras está sin pulsar.

Aunque se trate de un sensor muy sencillo puede resultar muy útil para ciertas aplicaciones como por ejemplo para saber si el robot tiene algo en sus brazos para cogerlo, para cerrar puertas, detectar choques, o para iniciar cualquier tipo de evento cuando se produzca la pulsación del sensor.

Funcionamiento

Como ya se ha comentando antes, el funcionamiento de este sensor es como el de un interruptor. Dispone de un muelle que mantiene separados los dos extremos. Si se ejerce presión sobre el muelle, éste se encogerá permitiendo que los dos extremos hagan contacto y permitan la circulación de la corriente.

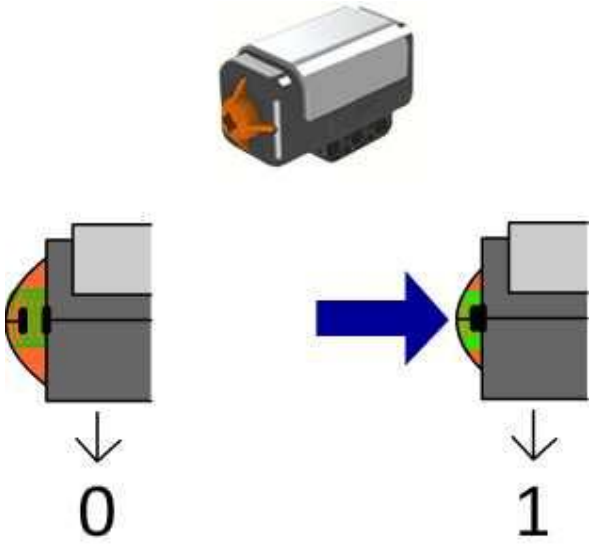


Figura 1: Funcionamiento del sensor de contacto.

Por lo tanto, si el sensor no está en contacto con nada devolverá un 0 mientras que si se encuentra en contacto con cualquier cosa devolverá un 1.

Nota: La superficie del interruptor es bastante pequeña, por lo que habrá que controlar bien donde colocamos el sensor según lo que se quiera detectar, ya que si quiere topar con objetos pequeños puede resultar complicado llegar a entrar en contacto con ellos. Otra opción es montar una "extensión" al sensor que incremente la superficie de contacto con el interruptor.

1.1. Actuadores

Los actuadores son los dispositivos que se añaden a la estructura y que dotan al robot de movimiento. De esta manera el robot será capaz de desplazarse, abrir o cerrar unas pinzas, girar, lanzar objetos,…

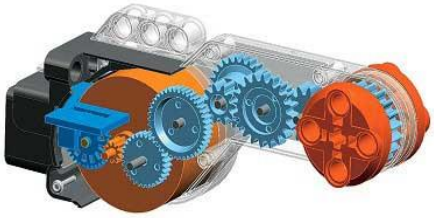
El kit de lego Mindstrom NXT 2.0, adquirido para la realización del proyecto, contiene 3 motores para acoplar al robot, o al mecanismo que se quiera componer.

1.1.1. Motor

El motor LEGO NXT es el motor específico de la línea NXT. El sensor de rotación, mide las rotaciones del motor en grados o rotaciones completas (con una exactitud de +/- un grado). Una rotación es igual a 360 grados.

El sensor de rotación también permite fijar distintas velocidades al motor, cambiando el parámetro potencia (power). También permite girar el motor unos grados determinados.

En la siguiente imagen se muestra el sistema de engranajes que incluye el motor.



En la siguiente tabla se muestran las características mecánicas y eléctricas del motor:

Motor NXT

Peso(gr)	80
Velocidad libre (RPM)	170
Consumo libre (mA)	60
Par motor (N/cm)	50
Consumo frenado (mA)	2000

1.3. Ladrillo NXT

El principal componente es un controlador con forma de ladrillo, denominado Ladrillo Inteligente NXT. Dicho elemento es el encargado de almacenar y gestionar los programas que se creen. Dicho controlador sería el cerebro del robot, la unidad central que gestiona todos los procesos y que se encarga de unificar entradas y salidas.

Este nuevo ladrillo inteligente programable, es el nuevo cerebro de los robots LEGO. Dispone de 4 entradas y 3 salidas. Los programadores pueden transmitir los datos con el ordenador a través de un cable USB (que se incluye en el kit) o por tecnología Bluetooth.

El ladrillo puede disponer de hasta 4 sensores y controlar hasta tres motores, por medio de tomas RJ12. Las cuales son muy similares, pero incompatibles, a las del cable telefónico.

La pantalla de la que dispone el ladrillo es una pantalla monocromática de 100x64 pixeles. Además dispone de 4 botones que pueden permitir la navegación por la interfaz de usuario mediante el sistema de menú jerárquico. También incorpora un altavoz con el qué se pueden reproducir archivos de sonidos con una frecuencia de muestreo de hasta 8 KHz.

El ladrillo permite ser programado desde el propio NXT, o bien desde el PC. Éste dispone de un microprocesador 32 bits y está preparado para controlar cualquier robot, o dispositivo, que se construya.

El ladrillo requiere el uso de 6 baterías AA (de 1,5 V cada una) o la batería Ion-Litio recargable 9798, para su funcionamiento.

Para la programación del ladrillo, Lego ha lanzado el ladrillo con un firmware y herramientas para desarrolladores con software abierto, junto con esquemas para todos los componentes de hardware.

Esto convierte al ladrillo de Lego NXT en un sistema de software abierto, aunque no se comercializa como tal.

Muchas de las herramientas para desarrolladores disponibles, que contienen documentación para NXT:

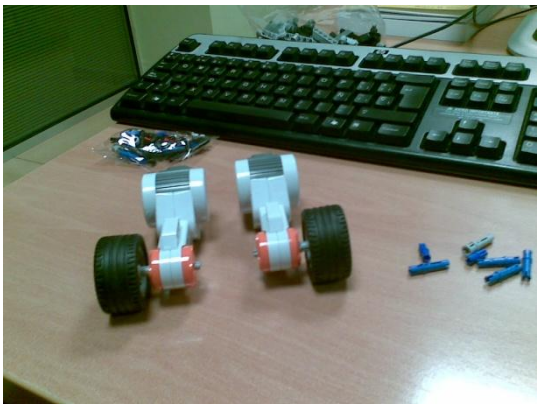
- SDK (Software Developer Kit) incluye información sobre los drivers del USB en el host, formato de archivo ejecutable y referencia de código de bytes.
- HDK (Hardware Developer Kit), que incluye documentación y esquemas para el ladrillo NXT y los sensores.
- BDK (Bluetooth Developer Kit), que incluye documentos de los protocolos usados en las comunicaciones por Bluetooth.

1.4. Montaje del robot

Desglose de todas las piezas que se incluyen en el kit Lego Mindstroms NXT 2.0, para el montaje de robot.

El primer paso consiste en preparar las ruedas que darán movilidad al robot. Para ello se necesitan dos de los tres motores que contiene el kit, dos ruedas, dos ejes número 7 y 4 topes para dichos ejes.

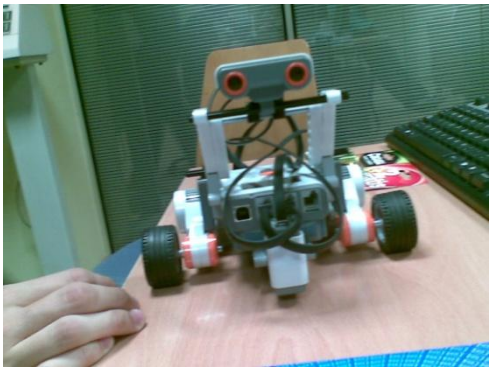
En el siguiente paso se acoplarán los dos motores al ladrillo.



Se conectan los motores en el NXT en los puertos B y C, mediante los cables RJ12.

A continuación se coloca la rueda trasera, qué servirá de estabilizadora para el movimiento, y también el sensor de color. El cual se conecta en el NXT a través del puerto 3.

Por último se conecta el sensor de ultrasonidos, qué servirá para detectar obstáculos, en el NXT mediante el puerto 4. El resultado de este último paso se refleja en las imágenes siguientes.



Por último se introducen las pilas en el ladrillo NXT. Y con esto se da por acabado el montaje del robot seguidor.

1.5. Instalación de Lejos en Windows usando Eclipse

Aquí explicaremos como instalar y configurar todo el software necesario para desarrollar programas en java para el robot Lego Mindstorms NXT. Pasos que hay que seguir:

1. Instalar Java en la computadora.

2. Instalar el driver Lego NXT USB en la computadora.

3. Configurar Eclipse en la computadora.

1. Instalar Java en la computadora

Descargar e instalar Java SE (Stadard Edition) JRE (Java Runtime Environment). Si ya se tiene instalado java, comprobar por lo menos que sea la versión 5. Luego se instalará Eclipse, que contiene todas las herramientas necesarias para escribir y compilar los programas.

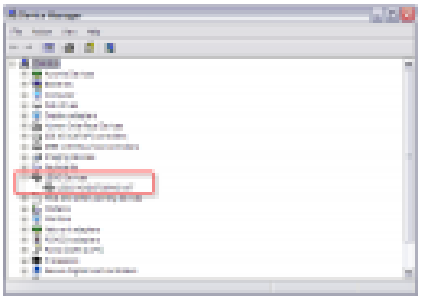
2. Instalar driver USB de Lego

El NXT se puede conectar a la computadora por USB o por Bluetooth. Primero se debe instalar el driver y después, conectar el NXT con el cable USB. No se necesita instalar el software que viene en el CD del robot porque no se programará en NXT-G (El software de lego basado en Labview). Solo se necesita instalar el driver USB que está disponible en el sitio oficial de Mindstorms. Si ya se ha instalado el software original de Mindstorms no es necesario desinstalarlo, solo revisar si el sitio de Mindstorms tiene alguna actualización para el driver USB.

Descargar el Driver de Mindstorms NXT. Descomprimir el archivo y ejecutar “setup.exe”. Es posible que Windows solicite reiniciar el equipo luego de la instalación. Luego, se conecta el NXT por medio del USB. Comprobar la correcta instalación del driver

comprobando si esta en el administrador de dispositivos. Para hacer esto, dar click derecho en "Equipo" y luego "Propiedades" →"Administrador de dispositivos".

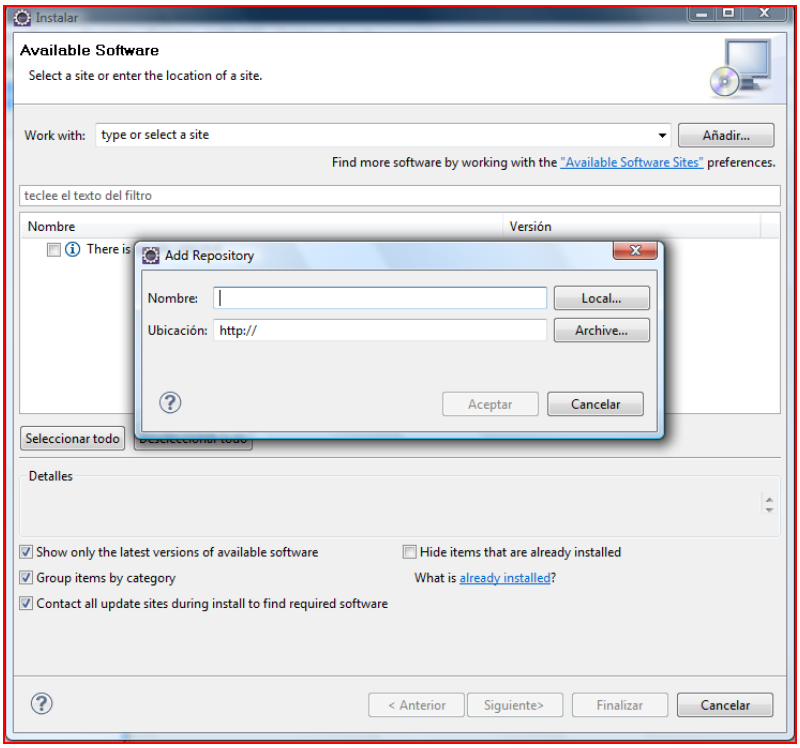
Tiene que aparecer "Lego Devices => Lego Mindstorms NXT".



3. Configurar Eclipse en la computadora

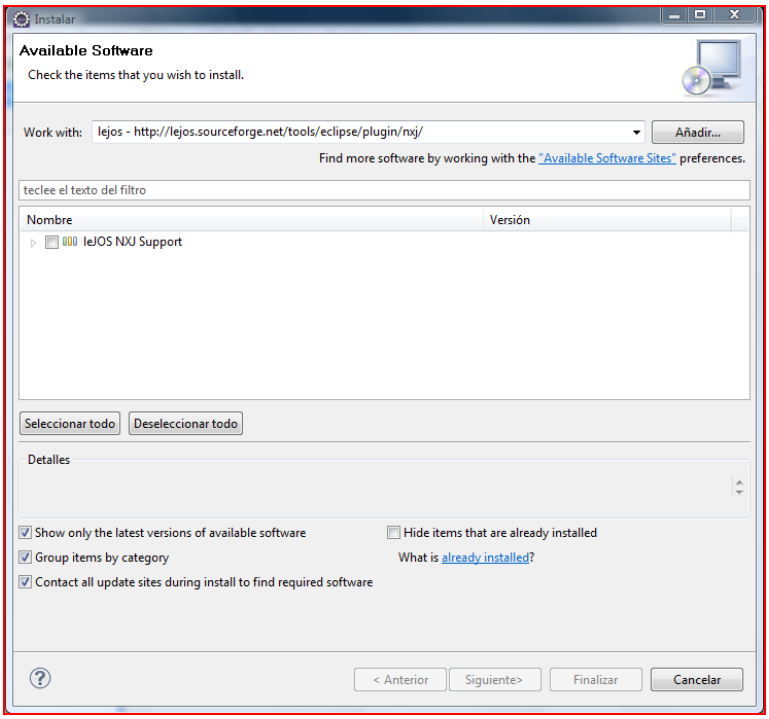
Instalación del plugin de eclipse

Una forma más fácil de crear un proyecto para LEJOS es usar el plugin de Eclipse. El plugin convertirá automáticamente sus proyectos en proyectos NXJ LEJOS. Para instalar el Lejos, haga clic en el menú Ayuda → Install New Software. En la siguiente pantalla:



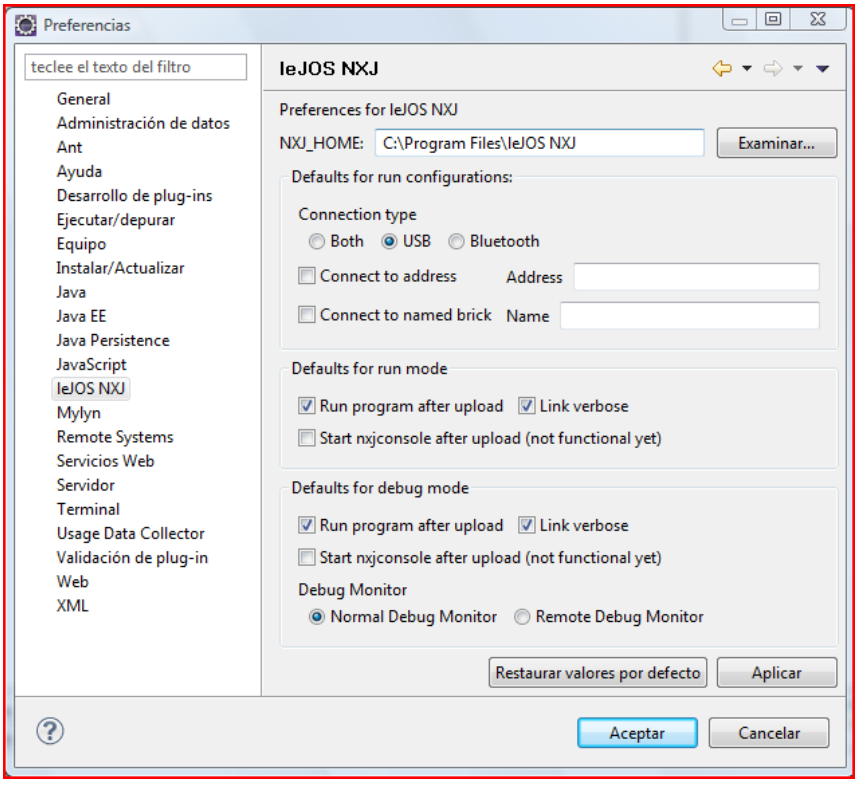
Seleccione “Añadir y se le pedirá una dirección. Escriba “http://lejos.sourceforge.net/tools/eclipse/plugin/nxj/”.

Aceptamos y en la siguiente pantalla marcamos la pestaña lejos. Pulsamos siguiente Eclipse buscara el plugin relacionado.



Pulse siguiente hasta aparezca la pestaña finalizar. El eclipse te pedirá que se reinicie.

Para Configurar el plugin en el eclipse ir a la pestaña ventana →Preferencias→examinar. Aquí das la ruta de donde está instalado lejos.

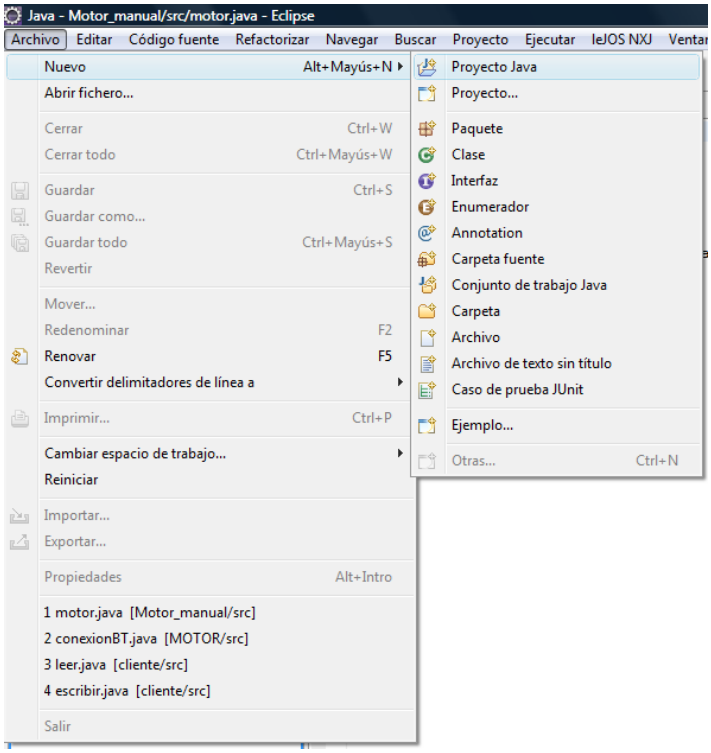


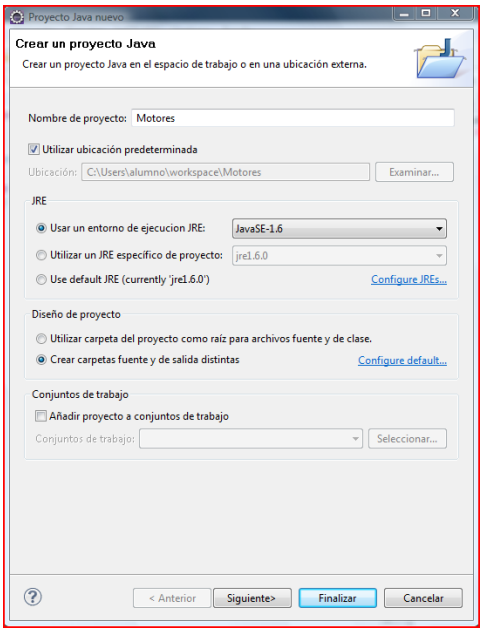
Cuando termine de configurar las preferencias, se hace click en “Apply” y después “Aceptamos”.

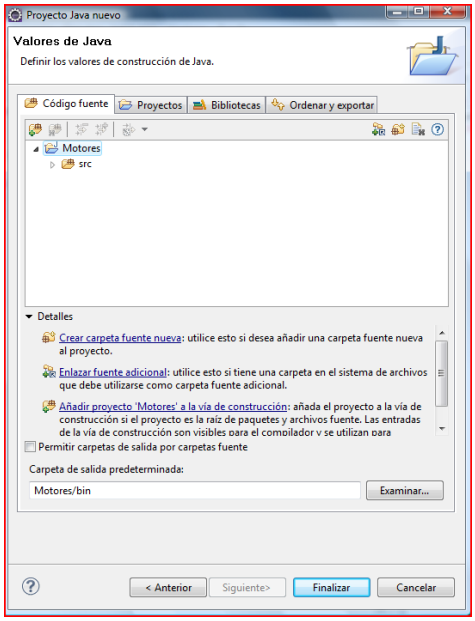
Modo de uso del pluging de Ecplise

Usted puede cargar el leJOS NXJ firmware hacia su NXT .Pulsando la pestaña lejos NXJ→ Upload firmaware.

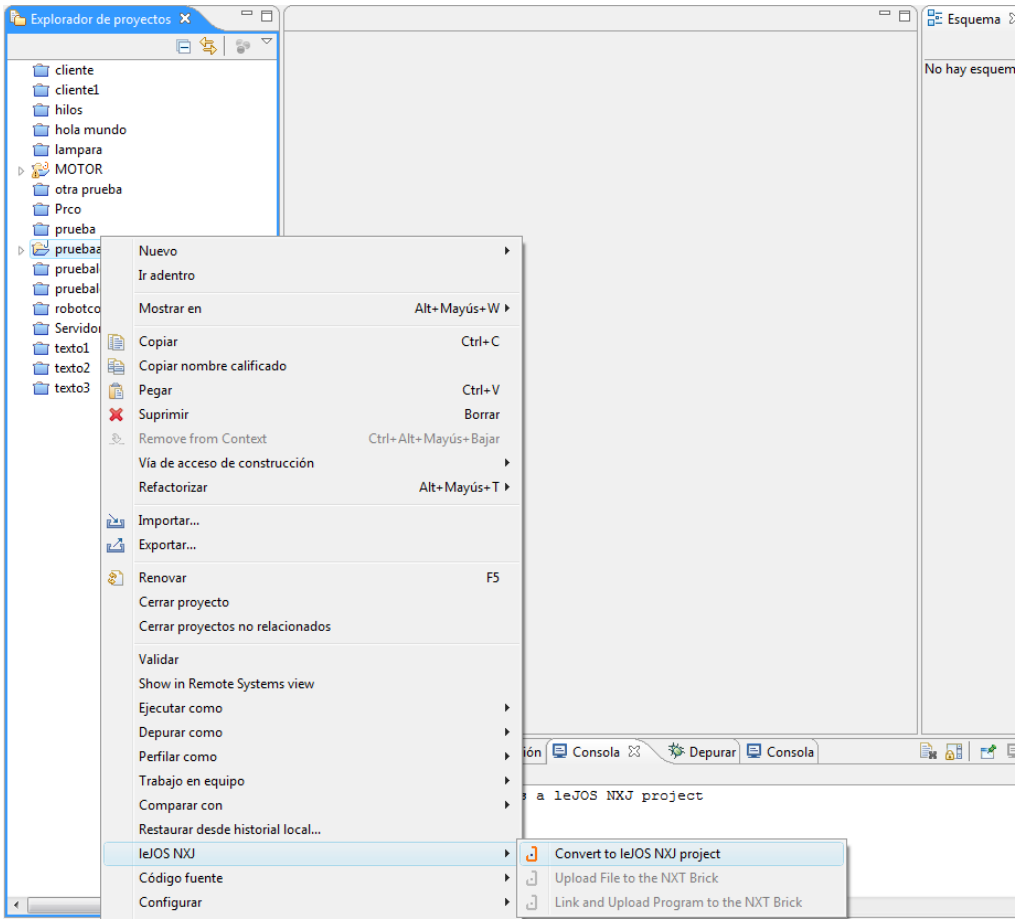
Para crear un proyecto nuevo leJOS NXJ usando el plugin, cree un proyecto Java.







Cuando su proyecto sea creado, Pulse el botón derecho del ratón sobre el proyecto, y seleccione “leJOS NXJ” y “Convert to leJOS NXJ Project”.



Esto marcara a su proyecto como un proyecto de leJOS NXJ y reemplazara el sistema de librerías JRE con las clases de su instalación de NXJ_HOME.

Ahora usted puede agregar paquetes y clases a su proyecto y compilarlo de la forma en la que usted lo haría para un proyecto de Java normal.

2. Primer programa

El primer programa que se va a realizar servirá para tener un primer contacto con el robot y todos sus componentes. Este primer programa tiene como objetivo final, que el robot pueda seguir una línea negra, de unos 2 centímetros de ancho. Además, con este primer programa, se pretende hacer una investigación que permita conocer el comportamiento de los distintos elementos que componen el robot, como son los sensores y los actuadores.

2.1. Introducción

Como se dijo anteriormente, el programa consistirá en conseguir que el robot sea capaz de seguir una línea de manera autónoma. Pero se tratará de conseguir este objetivo de la manera más simple posible, para luego ir complicándolo, añadiendo nuevos retos.

2.2. Objetivo

Conseguir que el robot avance un tiempo determinado y se pare. Una vez ha realizado esto, debe detectar si está en la posición correcta, y corregirla en caso de que no sea así.

2.3. Desarrollo

Para empezar, el trabajo se divide en tres partes fundamentales, localizar la posición correcta, el desplazamiento del robot y la conexión Robot-PC vía Bluetooth.

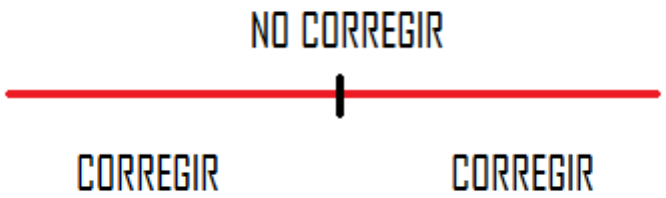
En lo relacionado a localizar la posición correcta, se presentaban dos opciones, o bien se buscaba que el robot se situara siempre sobre el color negro o bien se buscaba que el robot se situara el borde entre la línea negra y el fondo blanco.

El primer caso presentaba un problema, y era el de localizar por qué lado se salía el robot. Viendo que se trataba de un problema bastante complejo de atender, se optó por desecharlo y atacar el problema desde el punto de vista de la segunda opción, localizar el límite entre la línea y el fondo.

Para este caso había dos posibles soluciones, mantener el robot sobre el límite y que el robot corrigiera su posición cada vez que se desviara, o bien mantener el robot sobre un rango y que éste corrigiera su posición de acuerdo con unos criterios determinados.

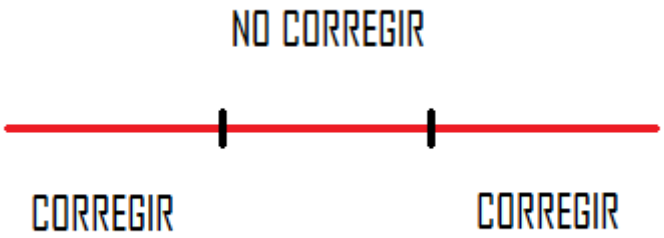
Gráficamente el resultado de las dos soluciones sería la siguiente:

1ª Opcion:



Como se puede apreciar en el esquema, cualquier movimiento del robot fuera del punto de control implicaría un proceso de corrección. Esto quiere decir que el robot va a carecer de fluidez en su movimiento.

2ª Opción:



Tal y como se observa en este nuevo esquema, el robot tiene más margen de maniobra, con lo que, mientras el robot se sitúe entre el valor máximo y mínimo, no hará falta corregirlo. Esto hará que el robot se mueva con mayor fluidez que con el sistema anterior.

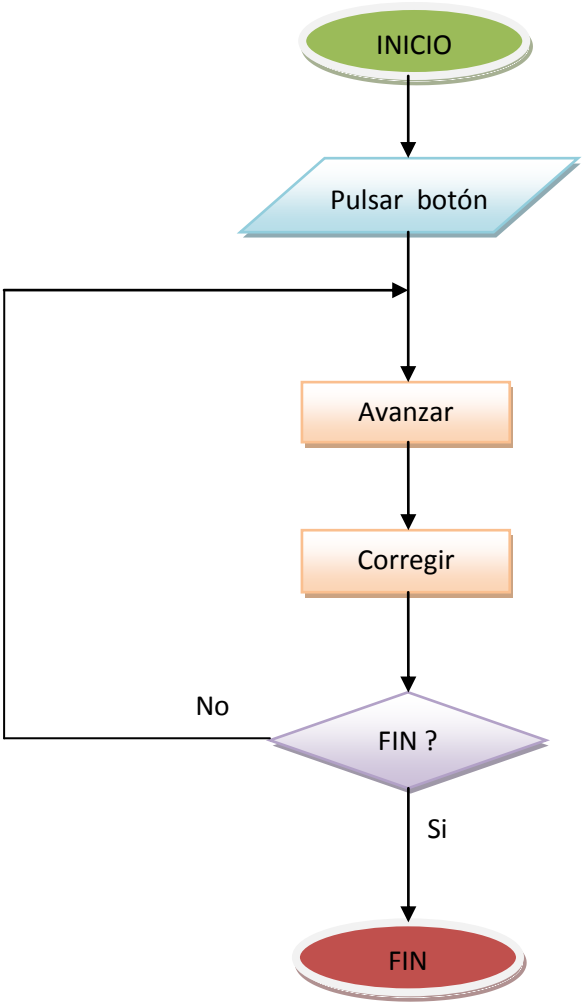
Es por ello que, para este primer objetivo de seguir la línea, se ha optado por la opción número 2.

2.4 Diagramas de flujo

El primer diagrama de flujo que se presenta, es el diagrama de flujo del programa principal. Comienza al producirse una condición de inicio, que puede ser al pulsar un botón.

Tras esto, se ejecuta la función de avance del robot y, tras un tiempo, se para. Una vez parado, se procede a acceder a la función de corregir, que se detalla más adelante.

Para terminar se comprueba la condición de parada, en caso de que sea correcta, el robot para, sino continúa con el proceso anterior.

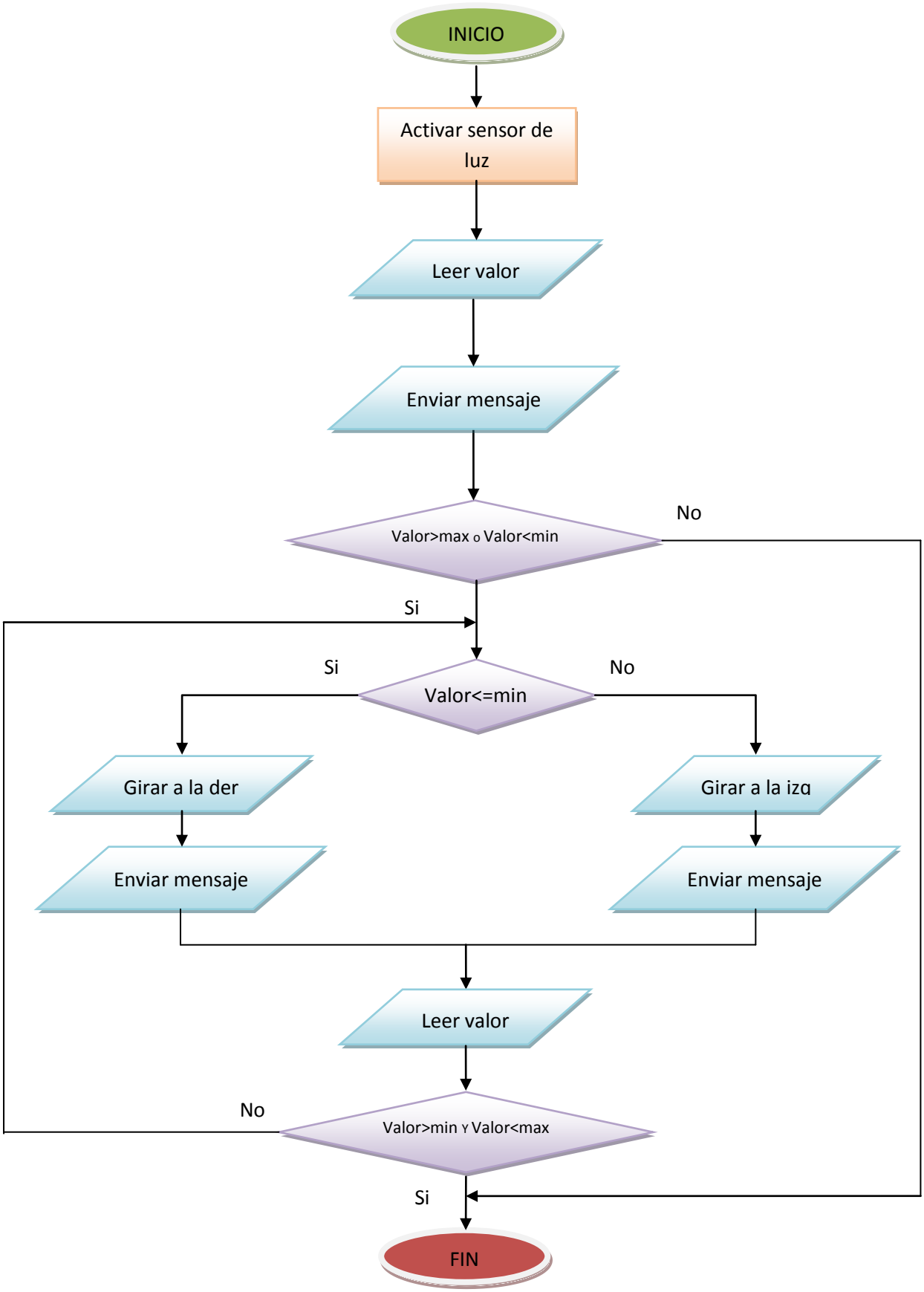


El diagrama que se presenta a continuación, es el de la función de corrección. Esta función es la encargada de devolver al robot a la posición correcta. Primero se activa el sensor, o se crea la variable. Tras esto último, se realiza una lectura del sensor y se envía, mediante la conexión Bluetooth, un mensaje al PC con dicho valor.

El siguiente paso será calcular si el valor se encuentra entre los márgenes que previamente se han marcado. Si el valor está dentro de ese margen, no se realiza ninguna corrección. De lo contrario, se analiza si el valor se encuentra por debajo del mínimo o por encima del máximo, actuando en consecuencia.

Como se puede observar en el esquema, si el valor se encuentra por debajo de mínimo, el robot entrará en la función girar a la derecha y, tras girar un número determinado de grados, enviará un mensaje al PC para advertir de dicho acto. En caso de que el valor sea mayor que el máximo determinado con anterioridad, el robot girará a la izquierda y enviará otro mensaje al PC informando sobre este giro.

Para terminar, se vuelve a leer el valor del sensor y se comprueba si el robot está dentro del rango determinado. Si la respuesta es que no, la función volverá a ejecutarse y volverá a corregir la posición. Si la respuesta es que si, entonces el programa saldrá de la función corregir.


INICIO
Activar sensor de luz
Leer valor
Enviar mensaje
Valor>max o Valor<min
No
Si
Si
Valor<=min
No
Girar a la der
Girar a la izq
Enviar mensaje
Enviar mensaje
Leer valor
No
Valor>min y Valor<max
Si
FIN

2.5. Clases definidas

En este apartado se determinarán las clases creadas para la realización del proyecto. Por un lado estará la clase que interviene en la parte del sensor de color, otra será la encargada del movimiento de los motores y la última será la que interviene en la comunicación PC-robot a través del Bluetooth.

2.5.1. Clase SensorLuz

La clase SensorLuz, es una clase que se crea con la intención de facilitar el uso del sensor de color que contiene el pack adquirido de Lego Mindstorm. Para la creación de la esta clase, se ha tenido en cuenta los posibles usos que se la van a dar a este sensor a lo largo del programa, o programas, que se ejecutarán con el robot.

Tras un primer estudio del sensor, se pudo observar que el sensor podía actuar de tres maneras diferentes. Por un lado como lámpara, activando sus LED´s de un color previamente determinado. Por otro lado, el sensor puede detectar el color que se sitúa a cierta distancia de él, así como detectar sus tres componentes (rojo, verde y azul). Y, por último, este sensor está preparado para detectar la concentración de luz ambiental.

Éste último será de gran utilidad para el proyecto de seguir la línea, pues permitirá establecer un rango sobre el cual se puede actuar.

Métodos

a) Devolver valor detectado (getValue)

Este método es el encargado de devolver un dato del tipo color, cada vez que se ejecuta. Dicho dato será el color detectado en ese mismo instante.

b) Devolver color detectado (getColor)

Con este método se pretende devolver un número entero que identifique un color determinado. Como base se ha tomado la tabla correspondiente de la API de Java, aunque el detector no es capaz de identificar todos los colores que ahí aparecen.

La tabla quedaría de la siguiente manera:

Identificador	Color
1	Negro
2	Azul
3	Cian
4	Gris oscuro
5	Gris
6	Verde
7	Gris claro
8	Magenta
9	Naranja
10	Rosa
11	Rojo
12	Blanco
13	Amarillo

c) Devolver nombre del color detectado (getColorName)

Este método establece la relación de la tabla anterior, con lo que dicho método devuelve una cadena con el nombre del color al que hace referencia el indicador que se le pase.

d) Devolver componente rojo (getColorR)

Este método se encarga de devolver un dato de tipo entero, en el que se encuentra reflejado el componente rojo del color detectado en ese instante.

e) Devolver componente verde (getColorG)

El método se encarga de devolver un dato de tipo entero, en el que se indica el componente verde del color detectado en ese instante.

f) Devolver componente azul (getColorB)

El método se encarga de devolver un dato de tipo entero, en el que se indica el componente azul del color detectado en ese instante.

g) Devolver brillo o luz ambiental(getBrillo)

Con este método se devuelve el valor normalizado del brillo de la luz blanca detectada.

Los valores bajos se interpretan como oscuros y los altos como claros. El rango de estos valores dependerá del dispositivo.

h) Establecer color(setColor)

Con este método, el sensor actuará como lámpara, encendiendo los LED´s de manera que entreguen una luz con el color indicado. Dicho color dependerá de las características del sensor.

El tipo de dato a introducir ha de ser entero, y el color se correspondería con los de la tabla siguiente:

Identificador	Color
7	Negro
2	Azul
12	Cian
11	Gris oscuro
9	Gris
1	Verde
10	Gris claro
4	Magenta
5	Naranja
8	Rosa
0	Rojo
6	Blanco
3	Amarillo
-1	Ninguno

i) Establecer nivel alto

Es un método que se podría emplear para calibrar el sensor, estableciendo, como nivel alto, el dato detectado en ese mismo instante.

j) Establecer nivel bajo

Es un método que se podría emplear para calibrar el sensor, estableciendo, como nivel bajo, el dato detectado en ese mismo instante.

Programa

```
import lejos.nxt.ColorSensor;
import lejos.nxt.SensorPort;
import lejos.robotics.Color;


public class SensorLuz {

      //Devuelve el color detectado, en formato color

    public Color getValue()
    {
        Color x;
        ColorSensor sensor= new ColorSensor(SensorPort.S3);

        x=sensor.getColor();

        return x;
    }



       //Devuelve un número entero de acuerdo con el color detectado


    public int getColor()
    {
        ColorSensor sensor= new ColorSensor(SensorPort.S3);
        int x =0, rtdo=-1;
        x=sensor.getColorID();

        if (x == 7) rtdo=1;
        if (x == 2) rtdo=2;
        if (x == 12) rtdo=3;
        if (x == 11) rtdo=4;
        if (x == 9) rtdo=5;
        if (x == 1) rtdo=6;
        if (x == 10) rtdo=7;
        if (x == 4) rtdo=8;
        if (x == 5) rtdo=9;
        if (x == 8) rtdo=10;
        if (x == 0) rtdo=11;
        if (x == 6) rtdo=12;
        if (x == 3) rtdo=13;

        return rtdo;
    }

    //Devuelve el nombre del color detectado

    public String getColorName(int v1)
    {
      String name="";

      if(v1==1)name="Negro";
      else
      if(v1==2)name="Azul";
      else
      if(v1==3)name="Cian";
```

```
    else
    if(v1==4)name="Gris oscuro";
    else
    if(v1==5)name="Gris";
    else
    if(v1==6)name="Verde";
    else
    if(v1==7)name="Gris claro";
    else
    if(v1==8)name="Magenta";
    else
    if(v1==9)name="Naranja";
    else
    if(v1==10)name="Rosa";
    else
    if(v1==11)name="Rojo";
    else
    if(v1==12)name="Blanco";
    else
    if(v1==13)name="Amarillo";
    else
    name="Fallo";

    return name;
}

//Devuelve el componente rojo del color detectado

public int getColorR()
{
    ColorSensor sensor= new ColorSensor(SensorPort.S3);

    return sensor.getColor().getRed();
}

//Devuelve el componente verde del color detectado

public int getColorG()
{
    ColorSensor sensor= new ColorSensor(SensorPort.S3);

    return sensor.getColor().getGreen();
}

//Devuelve el componente azul del color detectado

public int getColorB()
{
    ColorSensor sensor= new ColorSensor(SensorPort.S3);

    return sensor.getColor().getBlue();
}

//Método que devuelve el valor de luminosidad ambiental

public int getBrillo()
{
    ColorSensor sensor= new ColorSensor(SensorPort.S3);

    return sensor.getNormalizedLightValue();
}
```

```
    //Método que emplea el sensor como lámpara

    public void setColor(int v1)
    {

        ColorSensor sensor= new ColorSensor(SensorPort.S3);


        sensor.setFloodlight(v1);
    }

    //Método que sirve para calibrar el nivel alto del sensor

    public void setNivelAlto()
    {
       ColorSensor sensor= new ColorSensor(SensorPort.S3);

       sensor.calibrateHigh();

    }

    //Método que sirve para calibrar el nivel bajo del sensor

    public void setNivelBajo()
    {
       ColorSensor sensor= new ColorSensor(SensorPort.S3);

       sensor.calibrateLow();
    }


}
```

2.5.2. Clase Motores

La clase DifferentialPilot contiene métodos para controlar los movimientos del robot; desplazamiento hacia adelante o hacia atrás en línea recta o una trayectoria circular o girar a una nueva dirección. Esta clase sólo funciona con dos motores controlados de forma independiente para dirigir de forma diferencial, por lo que puede girar.

DifferentialPilot *conductor* = new DifferentialPilot(43.2f, 145.7f, Motor.*B*, Motor.*C*, true); Para el constructor DifferentialPilot es necesario saber el diámetro de las ruedas, distancia del eje y los puertos que están conectados los motores.

Métodos

a) Método rotarangulos.

Tienes que dar los grados que quieres que gire. Si los grados son positivos gira a la izquierda si los grados son negativos gira a la derecha.

b) Método Integer getGrados

Primero convierte los valores de tipo double en integer. Aquí se envía a la función del mensaje los grados que se va a girar.

c) Método avanza

Se tiene que dar la distancia que se quiere avanzar, en milímetros. Este método se encarga de hacer que los motores avancen para adelante, si el valor es negativo y avanza para atrás si el valor es positivo.

d) Método velocidad

Este método es el que establece la velocidad de rotación del motor con la que éste va a avanzar.

e) Método acelerar

Establece la aceleración de ambos motores.

f) Método parar

Este método se emplea para parar los motores.

g) Método integer GetEstado

Con este método se informa si el motor está parado o encendido. Se devuelve un uno si el motor está parado y se devuelve un dos si está encendido.

h) Método integer getavanza

Primero convierte double en integer. Devuelve la distancia que recorre el motor.

i) Método Integer delante_atras

Este método informa si avanza hacia adelante o hacia atrás.

Programa

```
import lejos.nxt.*;
import lejos.robotics.navigation.DifferentialPilot;
public class Impulsor {

static DifferentialPilot conductor = new DifferentialPilot(43.2f, 145.7f, Motor.B, Motor.C, true);
static Boolean activo;

        public void rotarangulos(double angle)
        {
                conductor.rotate(angle);
                //Si los grados son + gira a la izq sino a la derecha
        }

        public Integer getGrados(Double x)
        {
                //aqui convierte el double en integer.
                Integer retorno;
                retorno= x.intValue();
                activo = false;
                return retorno;//aqui envio los grados que giro.
        }

        public void avanza(double dista)
        {
                conductor.travel(dista);
                //para que avance recto tiene que se - y para vaya para atras tiene que ser +
```

```
}

public void acelerar(int accel)
{
        conductor.setAcceleration( accel);
        //Establece la aceleración de ambos motores.

}

public void velocidad(double veloz)
{
        conductor.setRotateSpeed(veloz);
        // establece la velocidad de rotación del robot
}

public void parar()
{
        conductor.stop();//aqui mandamos parar el motor.
        activo=true;
}

public Integer GetEstado()
{
        Integer estado;
        if ( activo=true)
        {
                estado = 1;//esta apagado
```

```
                }
                else
                {
                        estado = 2;//esta encedido
                }
                return estado;
                //aqui te devuelve el estado del motor si está apagado devuelve un 1 si esta encendido te devuelve2

        }
        public Integer getavanza(Double dista)
        {
                Integer distancia;
                distancia= dista.intValue();
                return distancia;
                //aqui convierte los double a integer y envia la distancia recorida.
        }
        public Integer delante_atras(double dista)
        {
                Integer estado;
                if (dista<0)
                {
                        estado = 3;//avanza adelante
                }
                else
                {
                        estado =4;//avanza atras
```

```
            }

            return estado;

        }

}
```

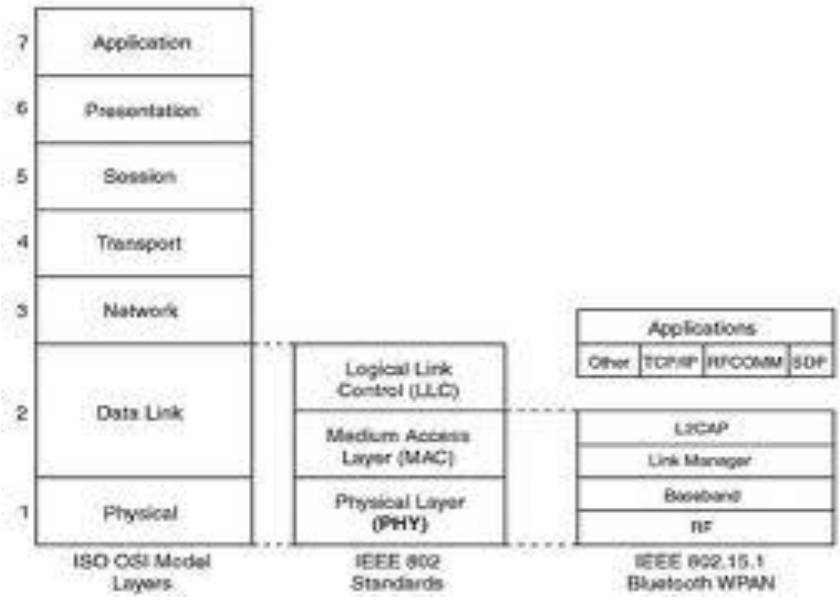
2.5.3. Clase BTConect

Manejo de Bluetooth en LeJOS

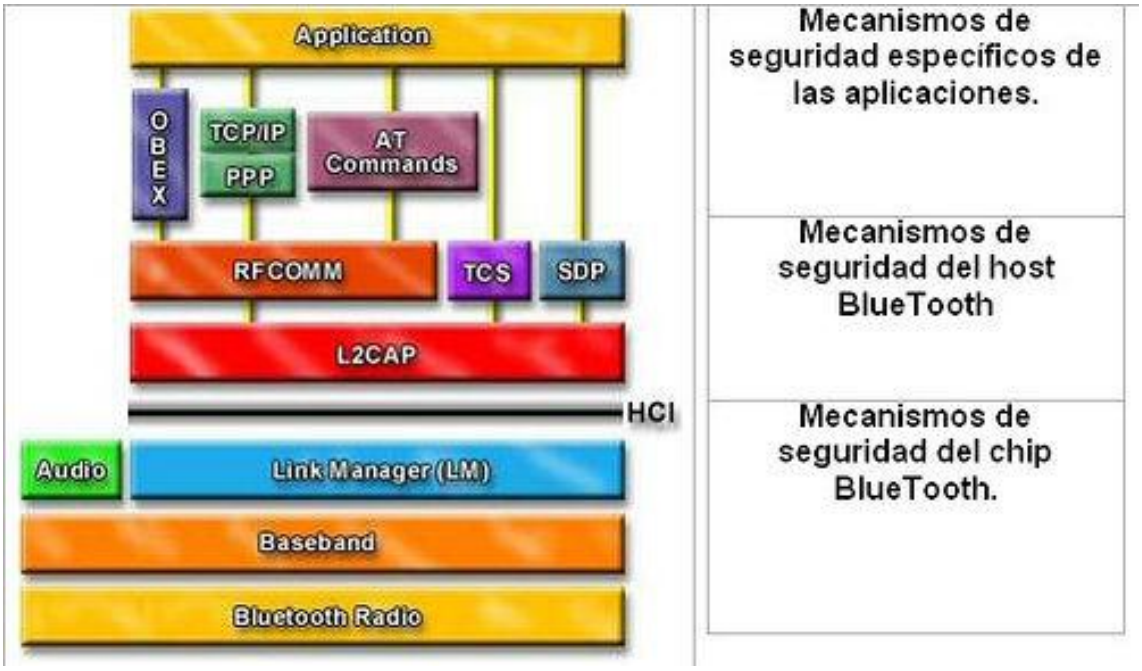
La conexión Bluetooth como la mayoría de comunicaciones en informática sirve para el intercambio de datos: el dispositivo que desea enviar los datos busca al destinatario, establece conexión con él, y va enviando conjuntos de información a través de un buffer (un canal de información).

El dispositivo que va a recibir los datos esta a la espera de recibir una petición de conexión, una vez aceptada y establecida la conexión irá leyendo del buffer dicha información. Una vez finalizado el proceso ambos deben cerrar la conexión, y el dispositivo de recepción quedará a la espera de una nueva conexión.

Arquitectura y protocolos de Bluetooth



Arquitectura Modelo OSI

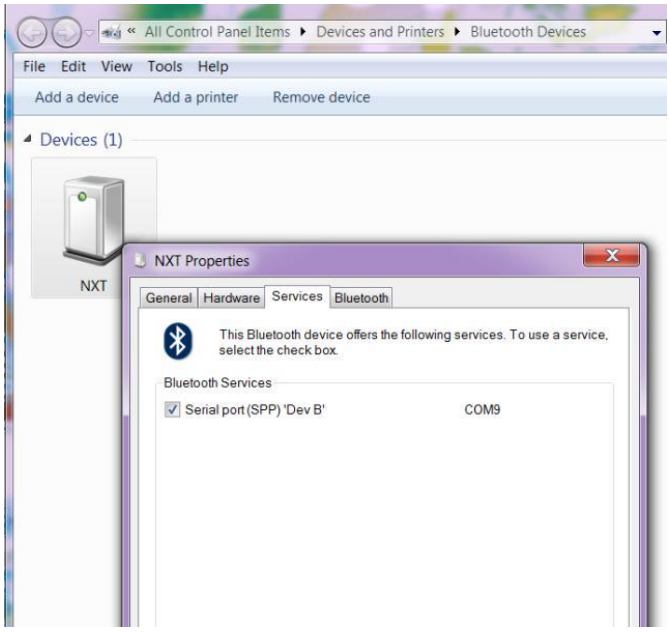


Protocolos del Bluetooth

Comunicación PC_NXT

Antes de utilizar el Bluetooth del NXT hay que asegurarse de que el ordenador que se va utilizar tiene esta posibilidad. Si no es el caso, debe utilizarse un Bluetooth dongle. Hay que asegurarse que se utiliza el adaptador correcto.

Para comunicar cualquier dispositivo Bluetooth con el ordenador necesitamos un adaptador Bluetooth para el ordenador (normalmente de tipo USB) sino lo tiene ya integrado el equipo.



Ahora agregaremos el Bluetooth del NXT a la lista de “Mis sitios Bluetooth”. Encendemos el NXT, hacemos doble click en el símbolo de Bluetooth de la barra de herramientas de Windows y en el menú que se despliega darle a “Agregar dispositivo Bluetooth”. Una vez lo haya encontrado lo seleccionamos, metemos la clave del NXT (por defecto trae 1234) y nos conectamos. Llegado a este paso ya estaremos listos para la comunicación entre el PC y el NXT.

El primer paso en la comunicación entre equipos es establecer una conexión. Una conexión es entre un PC (un iniciador) y un NXT (receptor). El receptor espera una conexión desde el iniciador. El iniciador se conecta a un dispositivo específico, el cual debe esperar una conexión. Cuando la conexión se ha establecido, se abren los canales de entrada y de salida de datos.

Los protocolos de comunicación en LeJOS entre PC y NXT se encuentran en una librería llamada *pccomm.jar* :

```
import lejos.pc.comm.*;
```

Ahora creamos dos clases que contendrán los métodos necesarios para la conexión por Bluetooth. Una será para el programa de PC y la otra para el NXT.

Clase BTConect (para PC)

```
void ConexionBTPC(){
      String nombre = "NXT"; //Nombre del Lego Mindstorms NXT 2.0
      String mac = "00165311E835"; //MAC del Lego Mindstorms NXT 2.0
      System.out.println(" Conectandose al " + nombre + mac);
      //La siguiente linea conecta los equipos por Bluetooth

      boolean conectado = conex.connectTo(nombre, mac,
NXTCommFactory.BLUETOOTH);
      if (!conectado) {
            System.err.println("Fallo al conectar al NXT");
            System.exit(1);
      }
}
```

En este método se conecta el PC al NXT, especificando el nombre del NXT al que se quiere conectar así como su MAC (líneas 2 y 3). Se realiza una salida por pantalla de los datos del NXT (línea 4). Tras esto, se crea la nueva conexión (línea 5), previamente se habrá definido (“NXTConnector conex;”).

Se guarda el estado de la conexión en un booleano (línea 6) para comprobar si la conexión fallo o no (línea 7).

```
void recibir(){
      try
      {
            mensajeNXT = entrada.readUTF();
            System.out.println(mensajeNXT);
      } catch (IOException e){
            System.out.println("Error de lectura" + e);
      }
}
```

Se lee el string del buffer de entrada de datos (línea 4) y se muestra por pantalla (línea 5). Si se produce un error en la lectura se manda un aviso por pantalla (línea 7).

```
void CerrarConex(){
              try {
                    entrada.close();
                    salida.close();
                    try {
                        Thread.sleep(100);
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
```

```
                conex.close();
            } catch (IOException e) {
                System.err.println("Fallo catch CerrarConex");
                e.printStackTrace();
            }
        }
```

El último método permite cerrar los canales de entrada y salida de datos (líneas 3 y 4). Luego se le da un tiempo para permitir vaciar el buffer (línea 6). Finalmente se cierra la conexión bluetooth (línea 10).

Clase BTConect (para NXT)

```
void conexionBT()
    {
            LCD.drawString(esperando,0,0);
            LCD.refresh();
            conex = Bluetooth.waitForConnection();
            LCD.clear();
            LCD.drawString(conectado,0,0);
            LCD.refresh();
    }
```

Método que muestra por la pantalla del NXT el mensaje de “Esperando” (línea 3). Luego está esperando la conexión por Bluetooth del otro dispositivo (línea 5). Una vez producida ésta, muestra el mensaje de “Conectado” en la pantalla del NXT (línea 7).

```
void conexionBT2(int tiempo, int modo) throws BluetoothStateException{
      LCD.drawString(esperando,0,0);
      LCD.refresh();
      conex = Bluetooth.waitForConnection(tiempo, modo);

      LCD.clear();
      LCD.drawString(conectado,0,0);
      LCD.refresh();
}
```

Funciona igual que el método anterior, solo que no se queda indefinidamente esperando por la conexión sino que espera un tiempo determinado en milisegundos, además podemos especificarle el modo exacto de conexión* (línea 4).

*Un tiempo igual a 0 es igual a esperar indefinidamente, los modos son RAW = 2, LCP = 1 o PACKET = 0. Para conectar a dispositivos no-NXT se suele usar PACKET.

```
void enviar(Integer codigo, Integer valor)
    {
      String mensaje;
      mensaje = codigo.toString() + valor.toString();
      try {
                  salida.writeUTF(mensaje);
                  salida.flush();
            } catch (IOException e) {
                  e.printStackTrace();
            }
    }
```

Este método recibe un código y un valor, y los pasa a un string (línea 4). Acto seguido lo envía por el canal de salida de datos (línea 6) y por si acaso se fuerza la salida de datos (línea 7).

```
void cerrar(){
      try {
                  //entrada.close();
                  salida.close();
                  try {
                        Thread.sleep(100);
                  } catch (InterruptedException e) {
                        e.printStackTrace();
                  }
                  conex.close();
                  LCD.clear();
            } catch (IOException e) {
                  e.printStackTrace();
            }
    }
```

El siguiente método permite cerrar los canales de entrada y salida de datos (líneas 3 y 4). Luego se le da un tiempo para permitir vaciar el buffer (línea 6). Y finalmente se cierra la conexión Bluetooth (línea 10).

```
    /*Devuelve true si el robot está conectado y false si no lo está*/
          public boolean getEstado(){

            boolean rdo=false;

            if(conex!=null ){

                  rdo=true;
                  LCD.clear();
                        LCD.drawString(conectado ,0,0);

                        try {
                              Thread.sleep(2000);
```

```
                        } catch (InterruptedException e) {
                              // TODO Auto-generated catch block
                              e.printStackTrace();
                        }
                  }
                  else{
                        rdo=false;
                        LCD.clear();
                              LCD.drawString(noconectado,0,0);

                              try {
                                    Thread.sleep(2000);
                              } catch (InterruptedException e) {
                                    // TODO Auto-generated catch block
                                    e.printStackTrace();
                              }
                  }


                  return rdo;
            }
```

Con este método se devuelve un valor de tipo booleano, que sirve para comprobar el estado de la conexión. Dicho método devolverá “true” si se ha establecido la conexión y un “false” en caso contrario.

```
    /*Devuelve true si el robot está conectado y false si no lo está*/
    boolean getEstado(){

      boolean rdo=false;

      if(conex!=null ){
            rdo=true;
            LCD.clear();
                  LCD.drawString(conectado ,0,0);
      }
      else{
            rdo=false;
            LCD.clear();
                  LCD.drawString(noconectado,0,0);
      }

      return rdo;
            }
```

2.6. Programa principal

```
import lejos.nxt.LCD;

public class Ppal {

      public static void main(String[] args) {
```

```
        /*Bucle que hace avanzar el robot y corrige posicion*/
        do{

              adelante();
              corregir();

        }while(!esRojo());

}

public static void adelante()
{
        double dista = -30;
        int accel = 900;
        Motores conductor = new Motores();

        LCD.clear();
        LCD.drawString("AVANZANDO" , 0, 1);

        conductor.acelerar(accel);
        conductor.avanza(dista);

}

public static void corregir()
{

         SensorLuz sensor= new SensorLuz();

         int valor=0;
         int max=460,min=300;


         do
         {

               valor= sensor.getBrillo();


         /*Bucle que corrige la posición del robot*/

               while((valor>max)||(valor<min)&&!esRojo())
               {


                     if (valor<=min)
                     {
                           girarDer();


                     }else
                     {
                           girarIzq();

                     }


                     valor= sensor.getBrillo();
```

```
                }

                LCD.clear();
                LCD.drawString("TODO BIEN" , 0, 1);


            }while((valor>max)||(valor<min));

        }

        /*Corrige hacia la derecha*/
        public static boolean esRojo()
        {
              boolean rdo=false;
              SensorLuz sensor= new SensorLuz();

              if (sensor.getColor()==11)
              {
                    rdo=true;
              }


              return rdo;
        }

        /*Corrige hacia la izquierda*/
        public static void girarIzq()
        {
              Motores conductor = new Motores();
              double angle=1;

              conductor.rotarangulos(angle);

        }

        /*Corrige hacia la derecha*/
        public static void girarDer()
        {
              Motores conductor = new Motores();
              double angle=-1;

              conductor.rotarangulos(angle);

        }
}
```

2.7. Programa en funcionamiento

A continuación se presenta un enlace directo a un video del robot siguiendo una línea, mediante el empleo del programa presentado en este apartado:

http://www.youtube.com/watch?v=zXFOe5E22_Q

3. Segundo programa(árbitro)

Para el segundo programa se pretende que el robot siga una línea, al igual que el objetivo marcado para el primer programa, pero incorporando una serie de mejoras que afecten a la fluidez del movimiento y a la optimización de los recursos.

3.1 Introducción.

Como se dice en las líneas anteriores, el objetivo del programa vuelve a ser que el robot siga una línea, pero con un cambio importante en el planteamiento de cómo atacar ese objetivo.

3.2 Objetivo

Se pretende alcanzar el objetivo marcado empleando las clases que se crearon para el primer programa, con algunas pequeñas modificaciones que se detallarán más adelante, y mediante el uso de nuevas herramientas.

Si bien en el primer programa se actuaba de una manera secuencial, primero se avanzaba un tiempo y luego se comprobaba si el robot estaba en la posición correcta para, a continuación, corregir la posible desviación. Ahora lo que se pretende es que primero el robot identifique automáticamente los límites sobre los que va a trabajar, en lugar de introducirlos a mano como se hacía en el programa anterior. Y además la otra gran mejora que se pretende introducir es la de que el sistema trabaje por un método parecido a las interrupciones, en las que el programa está a la espera de determinados datos y, dependiendo de dichos datos, ejecute unas rutinas u otras.

De esta manera lo que se pretende es dotar al programa de un sistema de interrupciones que dependen de unas prioridades. Dándole así mayor prioridad a las funciones relacionadas con la corrección de la posición, o las condiciones de parada de los motores, que a las de movimiento de avance, por ejemplo.

El sistema que se va a emplear para este cometido se denomina arbitraje, en el que distinguen dos partes el árbitro y los comportamientos. El comportamiento del programa está controlado por el árbitro, que es el encargado de gestionar todo lo relacionado con los comportamientos, lleva el control de todos ellos una vez definidos.

En cuanto a los comportamientos, hay que decir que son unos objetos que se componen de tres elementos: takeControl(), action() y suppress(). El primero de ellos devuelve "TRUE" si se dan las condiciones idóneas para que se ejecute el programa que previamente se le haya programado. Dichas condiciones son determinadas por el programador, con lo que hay determinados procesos que se pueden controlar y dar seguridad a los movimientos del robot. Por ejemplo que no avance si hay algún objeto delante.

El segundo elemento mencionado es "action()", que serán las líneas de código que se tienen que ejecutar si las condiciones determinadas se cumplen. Para el ejemplo anterior, si se detecta un objeto, en el elemento "action()" se introduciría el código necesario para que se paren las ruedas.

El tercero de los elementos es suppress(), que engloba el código que se quiere ejecutar cuando dicho comportamiento pierde el control, o lo que es lo mismo, las condiciones que se tienen que dar para su activación dejan de cumplirse. Para el ejemplo empleado antes, podría ponerse el código necesario para hacer sonar un "beep", indicando así que se va a reanudar la marcha, ya que no hay un objeto delante del robot.

3.3 Desarrollo

Conviene identificar claramente cuáles son los puntos más importantes de este nuevo objetivo y en qué va a afectar al programa anterior. Por un lado se tiene que estudiar el comportamiento del programa (clases y objetos) con los que ya se cuentan para poder hacer funcionar el arbitraje y, por otro lado cómo afecta la corrección progresiva.

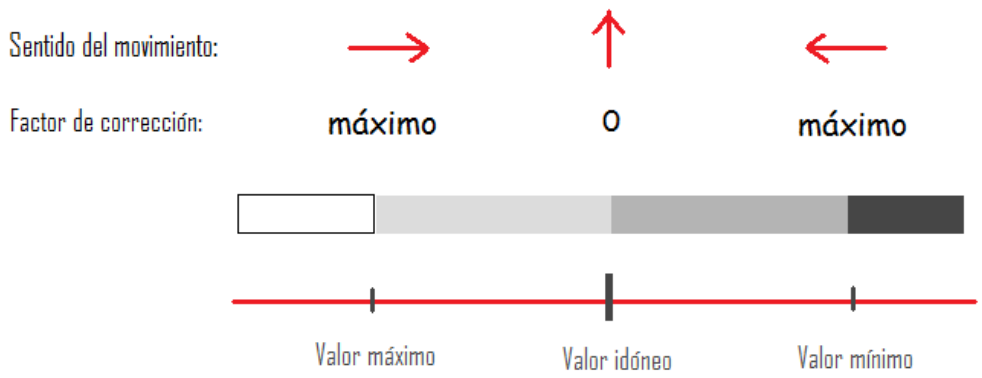
Una vez estudiado el control de los procesos por arbitraje, se decide dividir el trabajo en distintos procesos, identificando para cada uno de ellos las condiciones que

se tienen que dar para que, por un lado el movimiento sea fluido, y por otro lado sea eficiente.

Se distinguen dos procesos principales, el movimiento del robot y su parada, y las condiciones para cada uno de ellos. Las condiciones para el movimiento consisten en que no haya ningún objeto delante del robot, mientras que la condición de parada es la opuesta. Si se controla este hecho, no es posible un conflicto entre ambos comportamientos, si se diera el caso de la existencia de más comportamientos dentro del mismo programa, habría que dedicar tiempo suficiente para identificar las condiciones de toma de control para evitar conflictos o comportamientos indeseados.

En cuanto a la parte relacionada con el movimiento del robot, se vio la necesidad de incorporar una corrección progresiva, en lugar de aplicar el mismo ángulo de giro una vez que el robot abandonaba el área definida para su desplazamiento.

Para este tipo de corrección se opta por añadir un factor de corrección lineal que se suma, o se resta, a la potencia de cada rueda, de manera que una se moverá más rápido que la otra, y de manera proporcional a la diferencia entre el valor captado y el que se calcula como idóneo. El siguiente esquema representa el incremento del factor mencionado dependiendo de la posición del robot.



En la línea de abajo se representa la escala de valores entregado por el sensor según los colores que detecta. Los valor máximo y mínimo serían los calculados automáticamente y el valor idóneo sería la media calculada de ambos.

En la escala de colores, la inmediatamente superior a la explicada en el párrafo anterior, se representa de manera esquemática el área sobre la cual está situado el sensor del robot. Con el color negro se representa la línea a seguir, y con el color blanco el fondo sobre el que se dibuja la línea. Los colores grises representan los valores intermedios que capta el sensor.

En la parte “Factor de corrección” se pretende explicar, de una manera gráfica, como se pretende aplicar dicho factor para corregir la desviación del robot durante su movimiento. Como se puede observar en la imagen, el valor mínimo se corresponde con el cero, lo cual indica que las ruedas se mueven a la misma velocidad, lo que se traduce en un avance rectilíneo, indicado por la flecha superior del apartado “Sentido del movimiento”. A medida que el valor detectado por el sensor se aleja del valor idóneo, el factor de corrección se irá incrementando hasta alcanzar el máximo, valor que se alcanza al llegar al valor máximo o mínimo del rengo establecido. El movimiento del robot se corresponde con el representado por las flechas.

3.4 Diagramas de flujo

Los diagramas de flujo que se describen en este apartado se corresponden con las novedades incluidas para los nuevos objetivos. Las clases o las funciones empleadas en el este programa, y que se usaban en la versión anterior, no serán expuestas aquí porque ya se explicaron con detenimiento en los apartados anteriores, los correspondientes con el primer siguelíneas.

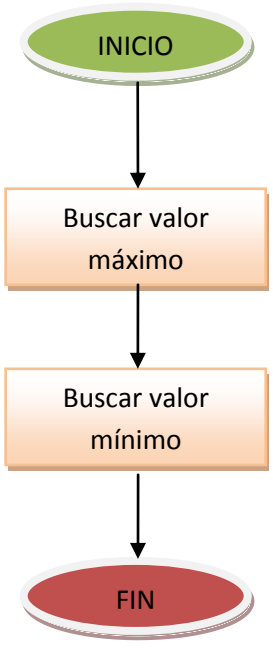
A continuación se presentan los esquemas correspondientes con la clase “ControBrillo” y el comportamiento “Movimiento”.

Clase ControlBrillo

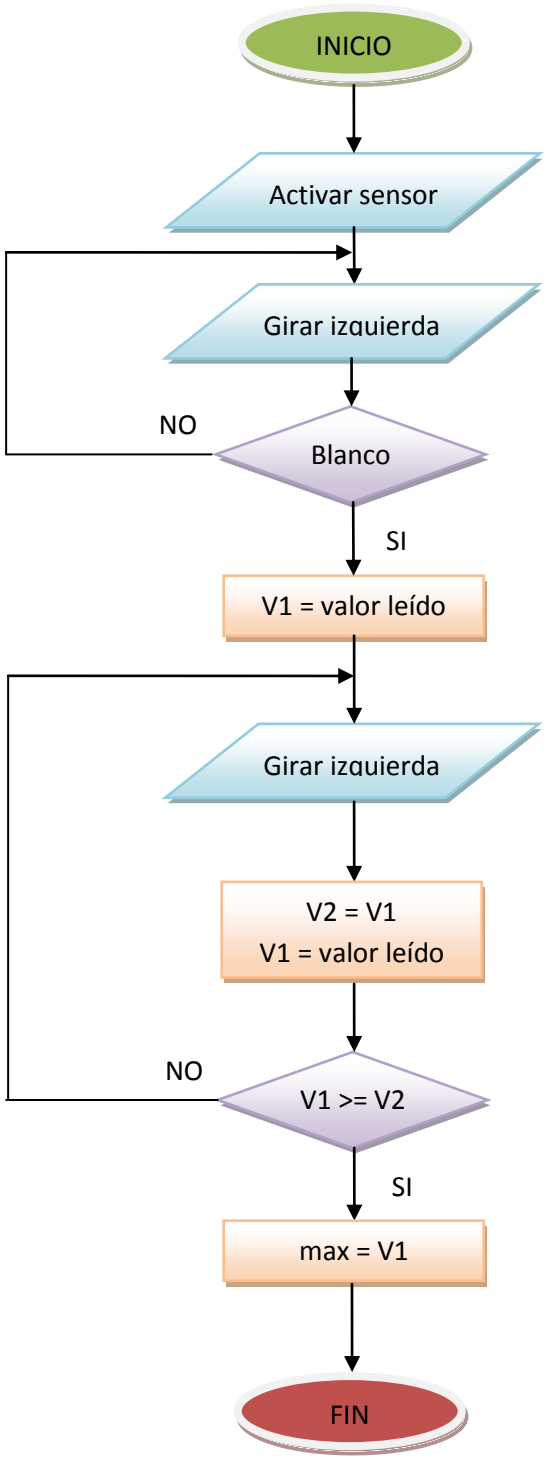
La clase "ControlBrillo" surge de la necesidad de que el robot, independientemente de cuál sea el lugar en el que se encuentra la línea a seguir, él pueda reconocer el rango en el que se va a mover. Se considerará que el robot está posicionado con la raya a su derecha, en el momento en el que se inicia el proceso.

Con esta función se pretende que el robot busque los márgenes máximo y mínimo de la luz ambiental, una vez éste haya sido colocado a la izquierda de la línea.

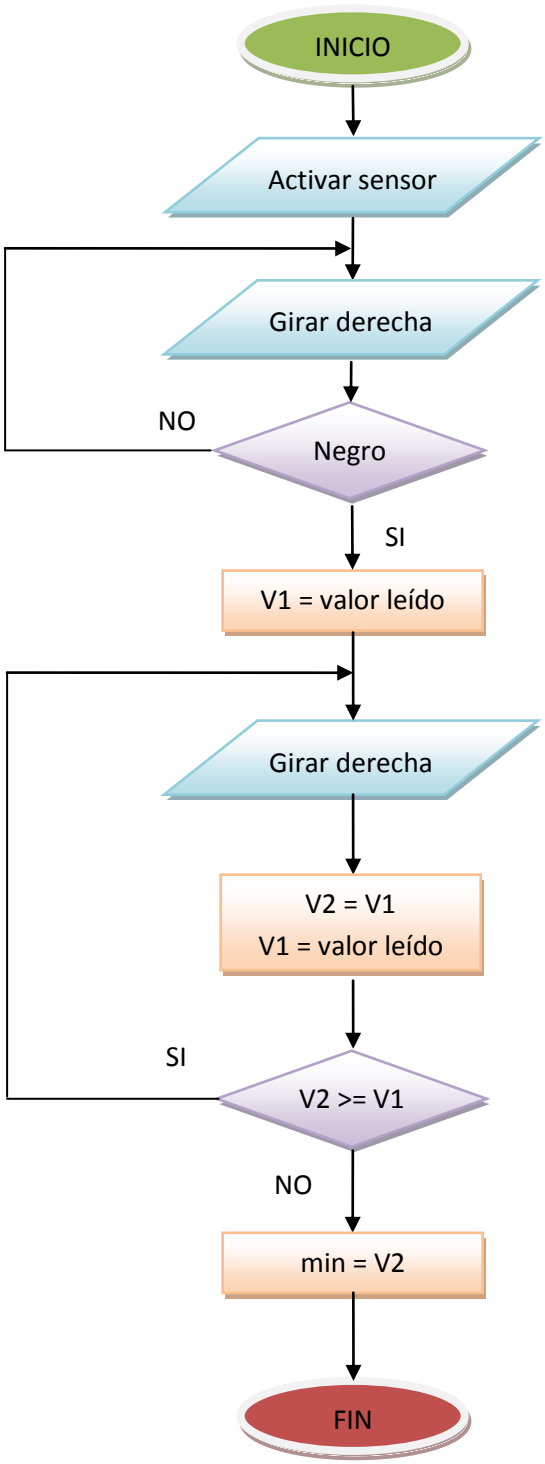
A continuación se presenta el esquema general de lo que se pretende que el robot realice al iniciarse el programa sigue líneas.



La función de búsqueda del valor máximo consiste en hacer que el robot se mueva hacia la izquierda sobre su propio eje hasta encontrar el color blanco y, una vez alcanzado siga girando, siempre que el valor detectado sea mayor o igual que el anteriormente registrado, manteniendo un margen que impida, en la medida de lo posible, conflictos con los valores recibidos por el sensor.



La función para la búsqueda del valor mínimo es la que se detalla a continuación. Consiste en hacer girar el robot hacia la derecha hasta encontrar el color negro y, a continuación, seguir girando sobre su propio eje (haciendo girar las ruedas a la misma velocidad, la misma distancia, pero en sentidos opuestos), hasta que el valor leído por el sensor sea menor o igual que el valor registrado con anterioridad, contando con un margen que solvente un posible conflicto con los valores registrados.



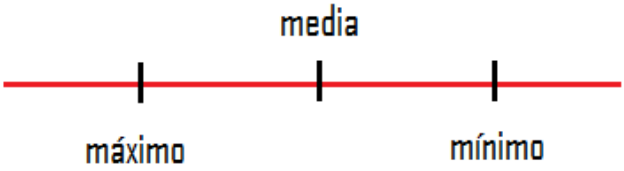
Los valores detectados se almacenarán en el objeto creado en el programa principal, destinado a albergar las variables que emplean el resto de los objetos del programa. Se trata de una clase nueva denominada "Contenedor", y que será el almacén de todas las variables que se van a emplear en todo el programa.

Clase Movimiento

Para este nuevo programa se ha optado por un proceso de corrección proporcional. Cuanto más se aleje el robot de lo que se considera que es la posición correcta, más será el gradiente de corrección aplicado.

El desplazamiento del robot en el primer programa estaba formado por dos procesos diferentes, el que se corresponde con el avance del robot y el que se corresponde con los giros a izquierda y derecha. Para el nuevo programa se ha optado por activar los motores por separado, a través de una función que permite configurar sentido de giro y potencia del motor. Con esto último, todo lo relacionado con el movimiento del robot queda relegado a una sola clase, clase Movimiento, que se encargará de entregar esos datos a la función.

En la siguiente figura se muestra el rango sobre el que se va a mover el robot para que realice la función de "siguelíneas".

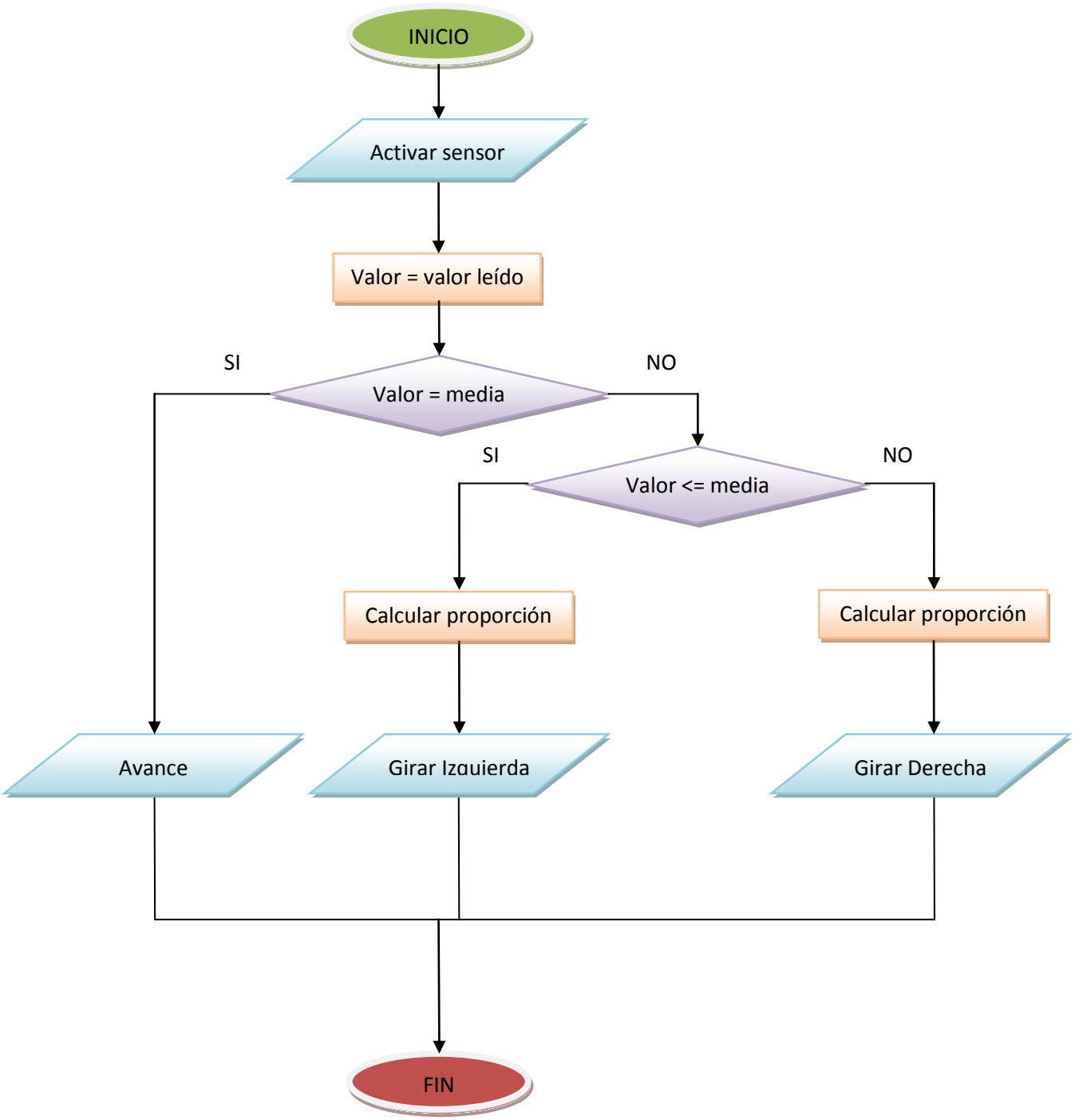


El proceso sería el siguiente, en un primer momento el robot identifica el valor máximo y mínimo que serían los límites sobre los que se va a desplazar. Una vez obtenidos esos valores, el robot calcula la media entre ambos, lo que se considera el valor perfecto, y es en ese punto donde las ruedas se desplazan a la misma velocidad, para hacer que el robot avance.

En caso de que el valor sea mayor que la media, se calcula cual es la diferencia entre el valor detectado y la media. Cuanto más se aproxime dicha diferencia al valor máximo permitido, mayor será la velocidad con la que la rueda izquierda avanza respecto de la derecha, lo que provoca un giro hacia la derecha.

Si por el contrario, el valor detectado es menor que la media, la velocidad de la rueda derecha avanzará más rápido respecto a la izquierda, también dependerá de la diferencia calculada entre ambos valores, cuanto más cercano esté el valor detectado del mínimo, mayor será la corrección.

El esquema resultante es el siguiente:



Con esto se consigue un movimiento más fluido y rápido, a pesar de que el movimiento del robot es corregido en cada ciclo, no como antes, que si el robot se encontraba dentro del margen no había corrección alguna.

3.5 Clases definidas

En este programa se emplean distintas clases, por un lado se encuentran algunas que ya se utilizaban en la versión anterior del siguelíneas, las cuales ya han quedado explicadas en los apartados correspondientes al anterior programa. Y por otro lado están las clases que se han creado específicamente para esta nueva versión y las nuevas especificaciones.

Las nuevas clases con ControlBrillo y Contenedor, y son las clases que se especifican a continuación.

3.5.1 Clase ControlBrillo

La clase ControlBrillo responde al esquema explicado en el apartado 3.4, denominado diagramas de flujo, y su función, tal y como se explica ahí, es la de determinar el rango de valores sobre los que se va a mover el robot.

Su código de programa es el siguiente:

```
public class ControlBrillo {

	/*DECLARACIÓN DE VARIABLES*/
	static VariablesCom var;
	static MotorPuerto motor;

	//Constructor
	public ControlBrillo(VariablesCom v1)
	{
		var = v1;
		motor=new MotorPuerto(var);
	}

	//Función que busca el valor máximo que lea el sensor de luz
	public  void buscarMax()
	{

		/*Declaración de variables*/
		SensorLuz sensor = new SensorLuz();
		int v1=0,v2=0;
```

```
            v1=sensor.getBrillo();     //se toma una primera muestra del
valor detectado

            //se gira a la derecha hasta encontrar color blanco
            while((sensor.getColor()!=12)){

                  girarDer();

            }

            //Ahora se gira a la derecha mientras el valor detectado
sea mayor o igual que el anterior(mas un margen)
            do{

                  girarDer();


                  v2=v1;
                  v1=sensor.getBrillo();

            }while(v1>=(v2+20));

            //se paran los motores
            parar();

            //se establece el valor máximo del rango
            var.setMax(v1);

      }

      //Función que busca el valor mínimo que lea el sensor de luz
      public  void buscarMin()
      {
            /*Declaración de variables*/
            SensorLuz sensor = new SensorLuz();
            int v1=0,v2=0;

            v1=sensor.getBrillo();  //se toma una primera muestra del
valor detectado

            //se gira a la izquierda hasta encontrar color negro
            while((sensor.getColor()!=1)){

                  girarIzq();

            }

            //Ahora se gira a la dizquierda mientras el valor detectado
sea menor o igual que el anterior(menos un margen)
            do{

                  girarIzq();


                  v2=v1;
                  v1=sensor.getBrillo();

            }while(v2>=(v1-10));
```

```
            //se paran los motores
            parar();

            //se establece el valor mínimo del rango
            var.setMin(v1);

      }

      //giro a la derecha sobre su eje
      private static void girarDer(){

            motor.calibrado(80,80,2,1);

      }
      //giro a la izquierda sobre su eje
      private static void girarIzq(){

            motor.calibrado(80,80,1,2);

      }

      //se paran los motores
      private static void parar(){

            motor.calibrado(0,0,3,3);

      }
      }
```

3.5.2 Clase Contenedor

La clase Contenedor es la encargada de albergar las variables que se vayan a emplear a lo largo del código. De esta manera, si algún objeto necesita conocer el valor máximo por ejemplo, deberá acudir a la variable determinada de la clase Contenedor y leerla.

El código de la clase Contenedor, para este programa es:

```
public class Contenedor {

      int max,min,media,distancia,brillo,powerB,powerC,modeB,modeC;
      boolean isCerca;

      public Contenedor (){

            max=0;
            min=0;
            media=0;
```

```
        distancia=0;
        isCerca=false;
        brillo=0;
        powerB=0;
        powerC=0;
        modeB=0;
        modeC=0;
}

public void setMax(int v1){

      max=v1;

}

public int getMax(){

      return max;

}

public void setMin(int v1){

      min=v1;

}

public int getMin(){

      return min;

}

private void setMedia(){

      media=((max-min)/2)+min;

}

public int getMedia(){

      setMedia();
      return media;

}

public void setBrillo(final int v1){

      brillo=v1;

}

public int getBrillo(){

      return brillo;

}
}
```

3.5.3 Clase MotorPuerto

En esta clase jugamos con la potencia de los motores. Cuando más potencia más velocidad, con menos potencia menos velocidad. Parámetros que utilizamos son:

1. Power ; que tiene que tener un valor de 0 a 100.

2. Modo; se define en base Puerto del motor.

   a. 1 = hacia delante
   b. 2 = hacia atrás
   c. 3 = paro
   d. 4 = float

Este método hace que el robot se desplace hacia delante.

```
public void adelante(int powerB,int powerC)
            {
             MotorPort.B.controlMotor(powerB, 1);
             MotorPort.C.controlMotor(powerC, 1);
            }
```

Este método hace que el robot se desplace atrás.

```
public void atras(int powerB,int powerC)
      {
       MotorPort.B.controlMotor(powerB, 2);
       MotorPort.C.controlMotor(powerC, 2);
      }
```

Este método hace que el robot se gire a la derecha. Para realizar el giro para el motor de la izq.

```
public void girarder(int powerB,int powerC)
      {
       MotorPort.B.controlMotor(powerB, 1);
       MotorPort.C.controlMotor(powerC, 3);
      }
```

Este método hace que el robot se gire a la izquierda. Para realizar el giro para el motor de la derecha.

```
public void girarizq(int powerB,int powerC)
      {
       MotorPort.B.controlMotor(powerB, 3);
       MotorPort.C.controlMotor(powerC, 1);
      }
```

Este método hace que el robot se gire a la izquierda, derecha, delante y atrás.

```
public void calibrado(int powerB,int powerC,int modeB,int modeC)
      {
       MotorPort.B.controlMotor(powerB, modeB);
       MotorPort.C.controlMotor(powerC, modeC);
      }
```

Este método actualiza los valores de la clase contenedor.

```
public void setpowerB(int powerB)
      {
       contenedor.powerB=powerB;

      }

public void setpowerC(int powerC)
      {
       contenedor.powerC=powerC;
      }

 public void setmodeB(int modeB)
      {
       contenedor.modeB=modeB;
      }

 public void setmodeC(int modeC)
      {
       contenedor.modeC=modeC;
       }
```

3.5.4 Clase SensorUltrasonidos

La clase ultrasonidos es la que se emplea para leer los valores entregados por el sensor de ultrasonidos.

En un principio parecía conveniente recibir dos datos, por un lado recibir la distancia a un objeto, la cual sería entregada en centímetros, y por otro lado, entregar un valor booleano que indique si existe un objeto próximo al robot.

Teniendo esto último en cuenta, se realizaron los métodos que se describen a continuación.

Métodos

a) Devolver el valor detectado (getDistancia)

El método getDistancia() se empleará para obtener el valor que detecta el sensor de ultrasonidos en dicho momento. Este método servirá para indicar, mediante una variable de tipo entero, la distancia en centímetros hasta un objeto.

b) Indicar si hay un objeto cerca(isCerca)

Este método se empleará para indicar, mediante el uso de una variable de tipo booleano, la presencia de algún objeto a una determinada distancia. Dicha distancia es preciso indicarla al hacer uso del método, introduciendo un número entero que indique la distancia a partir de la cual se quiere hacer constancia. Este método devolverá “true” si la distancia detectada es menor o igual que la indicada y “false”, si por el contrario, el valor detectado es mayor que el indicado.

Programa

```
import lejos.nxt.SensorPort;
import lejos.nxt.UltrasonicSensor;

public class SensorUltrasonidos {
```

```
UltrasonicSensor sensor =new UltrasonicSensor(SensorPort.S4);

public int getDistancia()
{

      return sensor.getDistance();

}

public Boolean isCerca(int v1)
{
      Boolean rdo=false;

      if (getDistancia()<= v1)
            rdo=true;
      else
            rdo=false;

      return rdo;
}

}
```

3.6 Comportamientos definidos

Los comportamientos son, como ya se dijo anteriormente, el código de programa que se quiere ejecutar cuando se cumplen unas determinadas condiciones, qué también serán definidas.

3.6.1 Comportamiento Obstáculo

El comportamiento denominado Obstáculo, es que en el que se insertará el código necesario para que el robot se pare de manera automática en el momento en el que detecta un obstáculo a través del sensor de ultrasonidos.

El código correspondiente a este comportamiento es el siguiente:

```
/*IMPORTAR LIBRERIAS*/
import lejos.nxt.LCD;
import lejos.robotics.subsumption.Behavior;
```

```
public class Obstaculo implements Behavior{

	/*SE DECLARAN LAS VARIABLES*/
	VariablesCom variables;
	MotorPuerto motor;


	//Constructor
	Obstaculo(VariablesCom v1)
	{
		variables=v1;
		motor = new MotorPuerto(variables);
	}

	public void action() {

		motor.calibrado(0, 0, 3, 3);  //Se para el motor

		//se imprime en la pantalla la palabra OBSTACULO
		LCD.clear(5);
		LCD.drawString("   OBSTACULO ", 0, 5);
		LCD.refresh();

		if(variables.distancia<10){  //El programa se termina al
detectarse un objeto a menos de 10 cm

			Ppal.enviarM(6,10); //Se envia mensage de final de
programa
			System.exit(0);

		}

	}

	public void suppress() {

	}

	public boolean takeControl() {

		return (variables.isCerca); //Si hay un objeto cercam, este
comportamiento toma el control

	}

	}
```

3.6.2 Comportamiento Movimiento

Este comportamiento es lo que permitirá al robot moverse con fluidez, al permitirle ir corrigiendo su posición de manera progresiva mientras se mueve. En los apartados anteriores, el destinado a desarrollo y el de diagramas de flujo, se explica la manera en la que funciona.

El código está pensado para establecer dos tipos de corrección, o bien se corrige el giro, especificando el ángulo y la velocidad, o bien se controlan el sentido de giro y la velocidad de las ruedas por separado. Es por ello que, dentro del código, se pueden observar cuatro funciones en lugar de dos. Pero las que se emplean son las relacionadas con el movimiento de las ruedas de manera independiente, porque se ha demostrado, tras unas pruebas, que el robot presenta un mejor comportamiento en el circuito.

A continuación se muestra el código empleado para su funcionamiento:

```
/*IMPORTAR LIBRERIAS*/
import lejos.nxt.LCD;
import lejos.robotics.subsumption.Behavior;


public class Movimiento implements Behavior{

	/*DECLARACION DE VARIABLES*/
	SensorLuz SensorL = new SensorLuz();
	MotorPuerto motor;
	VariablesCom var;

	//Constructor
	public Movimiento(VariablesCom v1){

		var = v1;
		motor = new MotorPuerto(var);
	}

	public void action() {

		correccion(var.getMax(),var.getMin(),var.getMedia());
//funcion de corrección del movimiento

	}

	public void suppress() {

	}

	public boolean takeControl() {

		return (!var.isCerca);  //Si no hay un objeto cerca, este
comportamiento toma el control

	}

	//Función de correción
	public void correccion(int v_max,int v_min,int v_media){

		/*Se declaran las variables*/
```

```
            int valor = var.getBrillo();
            int factor;

            /*Se imprime en la pantalla del NXT la palabra avanzando*/
            LCD.clear(5);
            LCD.drawString("   AVANZANDO", 0, 5);



            if(v_media == valor ){ //Si el valor detectado coincide con
la media las dos ruedas se mueven a la misma velocidad

                  motor.adelante(80, 80);

            }else{

                  if(valor<v_media){ //si el valor es menor que la
media

                        factor =
getRuedaIzqInt(valor,v_media,v_min,v_max); //se calcula el factor de
corrección

                        motor.calibrado(80-factor, 80+factor, 1, 1);

                  }else{ //si el valor es mayor que la media

                        factor =
getRuedaDerInt(valor,v_media,v_min,v_max); //se calcula el factor de
corrección

                        motor.calibrado(80+factor, 80-factor, 1, 1);

                  }

            }
      }

      public double getRuedaDer (int v,int v_media, int v_min){

            double rdo=0;
            double partes=0.0;

            if(v <= v_min){

                  rdo = 50;

            }else{

                  partes = (v_media-v_min)/50;

                  rdo = 50 - ((v - v_media)/partes);

            }

            rdo=rdo+50;

            return rdo;
      }

      public double getRuedaIzq (int v,int v_media, int v_max){
```

```
		double rdo=0;
		double partes=0.0;

		if(v >= v_max){

			rdo = 50;

		}else{

			partes = (v_max-v_media)/50;

			rdo = (v - v_media)/partes;

		}

		rdo=rdo+50;

		return rdo;
	}

	public int getGrados(int v,int v_media, int v_min){
		//int rdo=0;

		return 4;
	}
	/*-------------SIGUIENTE PRUEBA-------------*/

	public int getRuedaIzqInt (int v,int v_media, int v_min, int
v_max){

		int rdo=0;
		int partes=0;

		if(v <= v_min){

			rdo = 20;

		}else{

			partes = (v_media-v_min)/20;

			rdo = 20 - ((v - v_media)/partes);

		}


		return rdo;
	}

	public int getRuedaDerInt (int v,int v_media, int v_min, int
v_max){

		int rdo=0;
		int partes=0;

		if(v >= v_max){

			rdo = 20;

		}else{
```

```
                partes = (v_max-v_media)/20;

                rdo = (v - v_media)/partes;

            }


            return rdo;
        }
}
```

3.7 Hilos definidos

De la necesidad de obtener un sistema fluido, surge la idea de emplear hilos, o "Threads", para determinados procesos.

3.7.1 Hilo HiloUltraSon

El hilo HiloUltraSon, será el encargado de realizar mediciones a través del sensor de ultrasonidos, de manera periódica, para tener en cada momento la variable distancia actualizada, en el objeto de la clase Contenedor.

El código de este hilo es el siguiente:

```
import lejos.nxt.SensorPort;
import lejos.nxt.UltrasonicSensor;


public class HiloUltraSon extends Thread{

      /*Se declaran las variables*/
      VariablesCom variables;
      UltrasonicSensor sensor;

      //Constructor
      public HiloUltraSon(VariablesCom v1){
            sensor = new UltrasonicSensor(SensorPort.S4);
            variables=v1;
      }

      public void run() {
```

```
            while(true){

                  variables.distancia=sensor.getDistance(); //establece
el valor de distancia en el contenedor
                  variables.isCerca=cerca(variables.distancia);
//establece el valor del booleano que indica si hay algún objeto

            }
      }

      //Indica si el objeto está situado a menos de 20cm
      public boolean cerca(int v1){

            if(v1<=20) return true;
            else return false;
      }
}
```

3.7.2 Hilo HiloLuz

El hilo HiloLuz, será el encargado de realizar mediciones a través del sensor de color, de manera periódica, para tener en cada momento la variable brillo, del objeto de la clase contenedor, actualizada en todo momento.

El código correspondiente es el siguiente:

```
public class HiloLuz extends Thread{

      /*Se declaran variables*/
      VariablesCom var;
      SensorLuz sensor= new SensorLuz();

      //Constructor
      HiloLuz(VariablesCom v1){

            var=v1;
      }

      public void run(){

            do{

                  var.setBrillo(sensor.getBrillo()); //Se establece el
valor de brillo

            }while(1>0);
      }
      }
```

3.7.3 Hilo EnviarMen

El hilo "EnviarMen" es un hilo que tiene la función de enviar periódicamente mensajes al PC. De manera que en el PC se tendrá constancia en todo momento del estado en el que se encuentra el robot, y los valores registrados por los sensores.

Para enviar los mensajes se ha establecido un sistema de dos dígitos, tal y como se hizo en el primer programa, pero con algunos cambios. Dicho código es el siguiente:

Dígito 1	Dígito 2	Dato
1	Valor entero	Valor máximo
2	Valor entero	Valor mínimo
3	Valor entero	Valor media
4	Valor entero	Distancia detectada
5	Valor entero	Brillo detectado
6	10	Fin de programa

3.8 Programa principal

El programa principal será el espacio en el que se escribirá el código que crea los objetos que se van a emplear.

El código correspondiente es el siguiente:

```
import javax.bluetooth.BluetoothStateException;
import lejos.nxt.LCD;
import lejos.robotics.subsumption.Arbitrator;
import lejos.robotics.subsumption.Behavior;


public class Ppal {

	static BTConect conexion=new BTConect();
	static boolean conexOn=false;

	public static void main(String[] args) {
```

```
            /*-- DECLARACION DE VARIABLES --*/

            VariablesCom var = new VariablesCom();
            EnviarMen enviar = new EnviarMen(var);
            HiloLuz luz =new HiloLuz(var);
            Radar sonar = new Radar(var);
            HiloUltraSon sensorU = new HiloUltraSon(var);
            ControlBrillo rango = new ControlBrillo(var);
            boolean isOK = false;


            // Se espera conexion durante 5 segundos

            try {
                  conexion.conexionBT2(5000, 0);
            } catch (BluetoothStateException e) {

                  e.printStackTrace();
                  conexOn=false;
            }

            conexOn=conexion.getEstado();

            // Se establecen el valor máximo y el valor mínimo

            rango.buscarMax();
            rango.buscarMin();

            enviarM(1, var.max);
            enviarM(2, var.min);
            enviarM(3, var.getMedia());

            //Se imprimen en la pantalla del robot los valores maximo,
minimo y la media

            LCD.clear();
            LCD.drawString("Valor max:
"+Integer.toString(var.getMax()), 0, 0);
            LCD.drawString("Valor min:
"+Integer.toString(var.getMin()), 0, 1);
            LCD.drawString("Valor med:
"+Integer.toString(var.getMedia()), 0, 2);
            LCD.drawString("----------------------", 0, 3);

            //Se declaran los diferentes comportamientos

            Behavior b1 =   new Obstaculo(var);
            Behavior b2 =  new Movimiento(var);

            Behavior [] bArray = {b1, b2};
            Arbitrator arby = new Arbitrator(bArray,isOK);

          //Se arrancan los hilos y el árbitro

           enviar.start();
           luz.start();
             sensorU.start();
           sonar.start();
           arby.start();

      }
```

```
        public static void enviarM(int v1, int v2)
        {

                if (conexOn)
                {
                        conexion.enviar1(v1,v2);
                }
        }
}
```

3.9 Aplicación para PC

En este apartado se presenta el formulario que muestra los datos que entrega el robot durante su funcionamiento. El formulario se divide en tres partes: la dedicada a la comunicación, la dedicada al control del rango por el que se va a desplazar el robot y la dedicada al control de la presencia de obstáculos.

En la primera parte (indicada por el número uno en la figura) se tiene un botón, que será el que active el comienzo de la conexión NXT-PC, y además se dispone de un texto que, en caso de que exista una conexión entre el robot y el PC, advertirá de que la conexión ha sido realizada cambiando el mensaje por “conectado” en letras verdes.

En la segunda parte (la indicada por el número dos en la figura), una vez se establece la conexión, el robot envía los datos del calibrado, y seguidamente se muestran en pantalla. Estos datos vienen representados como máximo, mínimo y media.

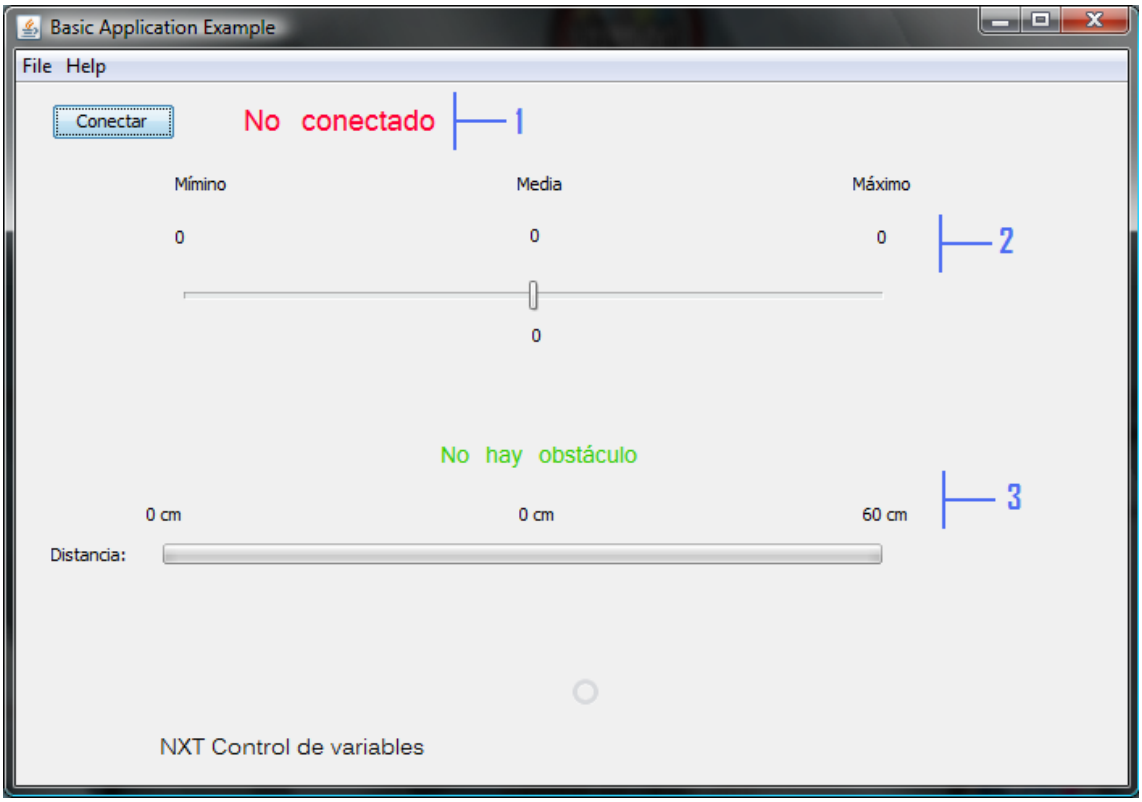
La barra que hay a continuación y el texto inferior sirven para representar el valor detectado por el sensor de color. La barra lo representa gráficamente y el texto numéricamente, de esta manera se visualiza la corrección que debe realizar el robot en cada momento.

Además se añaden dos textos con las palabras “IZQUIERDA” y “DERECHA”, que indican hacia donde gira el NXT en cada momento.

En el tercer área (la señalada por el número 3 en la figura), se encuentra localizado todo lo relacionado con el sensor de ultrasonidos. Primero hay un texto que advierte de la presencia de un obstáculo, si el sensor detecta que éste está a menos de 20 cm. El texto cambia su mensaje por “Obstáculo encontrado” con letras rojas.

A continuación se sitúa una barra que representa gráficamente la distancia al objeto. Si dicha distancia es superior o igual a 60 cm, la barra permanecerá con su valor al máximo.

La pantalla que se muestra al usuario se corresponde con la de la siguiente figura.



El código de programa correspondiente al formulario anterior es el que se muestra a continuación:

```
package pantallappal;

import java.awt.Color;
import org.jdesktop.application.Action;
import org.jdesktop.application.ResourceMap;
import org.jdesktop.application.SingleFrameApplication;
import org.jdesktop.application.FrameView;
import org.jdesktop.application.TaskMonitor;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.Timer;
import javax.swing.Icon;
import javax.swing.JDialog;
import javax.swing.JFrame;
```

```
/**
 * The application's main frame.
 */
public class PantallaPpalView extends FrameView {

   public PantallaPpalView(SingleFrameApplication app) {
      super(app);
      timer = new Timer (50, new ActionListener ()
{
   public void actionPerformed(ActionEvent e)
   {
      String men="";

          if(conexion.getEstado()){

            Lb_Conect.setForeground(Color.green);
            Lb_Conect.setText("Conectado");
            Lb_Conect.repaint();

       }else{

             Lb_Conect.setForeground(Color.red);
             Lb_Conect.setText("No hay conexión");
             Lb_Conect.repaint();



       }

      men=conexion.recibir();
      traduccion(men);
    }
});
      initComponents();

      // status bar initialization - message timeout, idle icon and busy animation, etc
      ResourceMap resourceMap = getResourceMap();
      int messageTimeout = resourceMap.getInteger("StatusBar.messageTimeout");
      messageTimer = new Timer(messageTimeout, new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            statusMessageLabel.setText("");
         }
      });
      messageTimer.setRepeats(false);
      int                                busyAnimationRate                                =
resourceMap.getInteger("StatusBar.busyAnimationRate");
```

```
        for (int i = 0; i < busyIcons.length; i++) {
            busyIcons[i] = resourceMap.getIcon("StatusBar.busyIcons[" + i + "]");
        }
        busyIconTimer = new Timer(busyAnimationRate, new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                busyIconIndex = (busyIconIndex + 1) % busyIcons.length;
                statusAnimationLabel.setIcon(busyIcons[busyIconIndex]);
            }
        });
        idleIcon = resourceMap.getIcon("StatusBar.idleIcon");
        statusAnimationLabel.setIcon(idleIcon);
        progressBar.setVisible(false);

        // connecting action tasks to status bar via TaskMonitor
        TaskMonitor taskMonitor = new TaskMonitor(getApplication().getContext());
        taskMonitor.addPropertyChangeListener(new
java.beans.PropertyChangeListener() {
            public void propertyChange(java.beans.PropertyChangeEvent evt) {
                String propertyName = evt.getPropertyName();
                if ("started".equals(propertyName)) {
                    if (!busyIconTimer.isRunning()) {
                        statusAnimationLabel.setIcon(busyIcons[0]);
                        busyIconIndex = 0;
                        busyIconTimer.start();
                    }
                    progressBar.setVisible(true);
                    progressBar.setIndeterminate(true);
                } else if ("done".equals(propertyName)) {
                    busyIconTimer.stop();
                    statusAnimationLabel.setIcon(idleIcon);
                    progressBar.setVisible(false);
                    progressBar.setValue(0);
                } else if ("message".equals(propertyName)) {
                    String text = (String)(evt.getNewValue());
                    statusMessageLabel.setText((text == null) ? "" : text);
                    messageTimer.restart();
                } else if ("progress".equals(propertyName)) {
                    int value = (Integer)(evt.getNewValue());
                    progressBar.setVisible(true);
                    progressBar.setIndeterminate(false);
                    progressBar.setValue(value);
                }
            }
        });
    }

    @Action
```

```
public void showAboutBox() {
    if (aboutBox == null) {
        JFrame mainFrame = PantallaPpalApp.getApplication().getMainFrame();
        aboutBox = new PantallaPpalAboutBox(mainFrame);
        aboutBox.setLocationRelativeTo(mainFrame);
    }
    PantallaPpalApp.getApplication().show(aboutBox);
}

/** This method is called from within the constructor to
 * initialize the form.
 * WARNING: Do NOT modify this code. The content of this method is
 * always regenerated by the Form Editor.
 */
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    mainPanel = new javax.swing.JPanel();
    Bt_Conect = new javax.swing.JButton();
    Lb_Conect = new javax.swing.JLabel();
    jSlider1 = new javax.swing.JSlider();
    Lb_max = new javax.swing.JLabel();
    Lb_media = new javax.swing.JLabel();
    Lb_min = new javax.swing.JLabel();
    Pb_dist = new javax.swing.JProgressBar();
    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    Lb_minV = new javax.swing.JLabel();
    Lb_maxV = new javax.swing.JLabel();
    Lb_distV = new javax.swing.JLabel();
    Lb_mediaV = new javax.swing.JLabel();
    Lb_brilloV = new javax.swing.JLabel();
    Lb_corrIzq = new javax.swing.JLabel();
    Lb_corrDer = new javax.swing.JLabel();
    Lb_obj = new javax.swing.JLabel();
    menuBar = new javax.swing.JMenuBar();
    javax.swing.JMenu fileMenu = new javax.swing.JMenu();
    javax.swing.JMenuItem exitMenuItem = new javax.swing.JMenuItem();
    javax.swing.JMenu helpMenu = new javax.swing.JMenu();
    javax.swing.JMenuItem aboutMenuItem = new javax.swing.JMenuItem();
    statusPanel = new javax.swing.JPanel();
    statusMessageLabel = new javax.swing.JLabel();
    statusAnimationLabel = new javax.swing.JLabel();
    progressBar = new javax.swing.JProgressBar();
    jLabel4 = new javax.swing.JLabel();
```

```
        jFrame1 = new javax.swing.JFrame();

        mainPanel.setName("mainPanel"); // NOI18N
        mainPanel.addComponentListener(new java.awt.event.ComponentAdapter() {
            public void componentShown(java.awt.event.ComponentEvent evt) {
                mainPanelComponentShown(evt);
            }
        });

        org.jdesktop.application.ResourceMap resourceMap =
org.jdesktop.application.Application.getInstance(pantallappal.PantallaPpalApp.class).g
etContext().getResourceMap(PantallaPpalView.class);
        Bt_Conect.setText(resourceMap.getString("Bt_Conect.text")); // NOI18N
        Bt_Conect.setName("Bt_Conect"); // NOI18N
        Bt_Conect.addMouseListener(new java.awt.event.MouseAdapter() {
            public void mouseClicked(java.awt.event.MouseEvent evt) {
                Bt_ConectMouseClicked(evt);
            }
        });

        Lb_Conect.setFont(resourceMap.getFont("Lb_Conect.font")); // NOI18N
        Lb_Conect.setForeground(resourceMap.getColor("Lb_Conect.foreground")); //
NOI18N
        Lb_Conect.setText(resourceMap.getString("Lb_Conect.text")); // NOI18N
        Lb_Conect.setName("Lb_Conect"); // NOI18N

        jSlider1.setName("jSlider1"); // NOI18N

        Lb_max.setText(resourceMap.getString("Lb_max.text")); // NOI18N
        Lb_max.setName("Lb_max"); // NOI18N

        Lb_media.setText(resourceMap.getString("Lb_media.text")); // NOI18N
        Lb_media.setName("Lb_media"); // NOI18N

        Lb_min.setText(resourceMap.getString("Lb_min.text")); // NOI18N
        Lb_min.setName("Lb_min"); // NOI18N

        Pb_dist.setMaximum(60);
        Pb_dist.setName("Pb_dist"); // NOI18N

        jLabel1.setText(resourceMap.getString("jLabel1.text")); // NOI18N
        jLabel1.setName("jLabel1"); // NOI18N

        jLabel2.setText(resourceMap.getString("jLabel2.text")); // NOI18N
        jLabel2.setName("jLabel2"); // NOI18N

        jLabel3.setText(resourceMap.getString("jLabel3.text")); // NOI18N
```

```
        jLabel3.setName("jLabel3"); // NOI18N

        Lb_minV.setText(resourceMap.getString("Lb_minV.text")); // NOI18N
        Lb_minV.setName("Lb_minV"); // NOI18N

        Lb_maxV.setText(resourceMap.getString("Lb_maxV.text")); // NOI18N
        Lb_maxV.setName("Lb_maxV"); // NOI18N

        Lb_distV.setText(resourceMap.getString("Lb_distV.text")); // NOI18N
        Lb_distV.setName("Lb_distV"); // NOI18N

        Lb_mediaV.setText(resourceMap.getString("Lb_mediaV.text")); // NOI18N
        Lb_mediaV.setName("Lb_mediaV"); // NOI18N

        Lb_brilloV.setText(resourceMap.getString("Lb_brilloV.text")); // NOI18N
        Lb_brilloV.setName("Lb_brilloV"); // NOI18N

        Lb_corrIzq.setFont(resourceMap.getFont("Lb_corrIzq.font")); // NOI18N
        Lb_corrIzq.setText(resourceMap.getString("Lb_corrIzq.text")); // NOI18N
        Lb_corrIzq.setName("Lb_corrIzq"); // NOI18N

        Lb_corrDer.setFont(resourceMap.getFont("Lb_corrDer.font")); // NOI18N
        Lb_corrDer.setText(resourceMap.getString("Lb_corrDer.text")); // NOI18N
        Lb_corrDer.setName("Lb_corrDer"); // NOI18N

        Lb_obj.setFont(resourceMap.getFont("Lb_obj.font")); // NOI18N
        Lb_obj.setForeground(resourceMap.getColor("Lb_obj.foreground")); // NOI18N
        Lb_obj.setText(resourceMap.getString("Lb_obj.text")); // NOI18N
        Lb_obj.setName("Lb_obj"); // NOI18N

        javax.swing.GroupLayout mainPanelLayout = new
javax.swing.GroupLayout(mainPanel);
        mainPanel.setLayout(mainPanelLayout);
        mainPanelLayout.setHorizontalGroup(

mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(mainPanelLayout.createSequentialGroup()
                .addGap(312, 312, 312)
                .addComponent(Lb_distV)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
181, Short.MAX_VALUE)
                .addComponent(jLabel1)
                .addContainerGap(126, Short.MAX_VALUE))
            .addGroup(mainPanelLayout.createSequentialGroup()
                .addGap(81, 81, 81)
                .addComponent(jLabel2)
                .addContainerGap(566, Short.MAX_VALUE))
```

```
            .addGroup(mainPanelLayout.createSequentialGroup()
                .addGap(22, 22, 22)
                .addComponent(Bt_Conect)
                .addGap(42, 42, 42)
                .addComponent(Lb_Conect)
                .addContainerGap(409, Short.MAX_VALUE))
            .addGroup(mainPanelLayout.createSequentialGroup()
                .addGap(91, 91, 91)
                .addComponent(Pb_dist,    javax.swing.GroupLayout.PREFERRED_SIZE,    446,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(132, Short.MAX_VALUE))
            .addGroup(mainPanelLayout.createSequentialGroup()
                .addGap(22, 22, 22)
                .addComponent(jLabel3)
                .addContainerGap(600, Short.MAX_VALUE))
            .addGroup(mainPanelLayout.createSequentialGroup()
                .addGap(99, 99, 99)
                .addComponent(jSlider1,    javax.swing.GroupLayout.PREFERRED_SIZE,    444,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(126, Short.MAX_VALUE))
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
mainPanelLayout.createSequentialGroup()
                .addContainerGap(456, Short.MAX_VALUE)
                .addComponent(Lb_corrDer)
                .addGap(213, 213, 213))
            .addGroup(mainPanelLayout.createSequentialGroup()
                .addGap(181, 181, 181)
                .addComponent(Lb_corrIzq)
                .addContainerGap(488, Short.MAX_VALUE))
            .addGroup(mainPanelLayout.createSequentialGroup()
                .addGap(320, 320, 320)
                .addComponent(Lb_brilloV)
                .addContainerGap(343, Short.MAX_VALUE))
            .addGroup(mainPanelLayout.createSequentialGroup()

.addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING, false)
                    .addGroup(mainPanelLayout.createSequentialGroup()
                        .addGap(99, 99, 99)
                        .addComponent(Lb_minV))
                    .addGroup(mainPanelLayout.createSequentialGroup()
                        .addContainerGap(99, Short.MAX_VALUE)
                        .addComponent(Lb_min)))

.addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
                    .addGroup(mainPanelLayout.createSequentialGroup()
```

```
                .addGap(180, 180, 180)
                .addComponent(Lb_media))
            .addGroup(mainPanelLayout.createSequentialGroup()
                .addGap(188, 188, 188)
                .addComponent(Lb_mediaV)))
        .addGap(180, 180, 180)

.addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
CENTER, false)
            .addComponent(Lb_maxV)
            .addComponent(Lb_max))
        .addGap(114, 114, 114))
      .addGroup(mainPanelLayout.createSequentialGroup()
        .addGap(263, 263, 263)
        .addComponent(Lb_obj)
        .addContainerGap(284, Short.MAX_VALUE))
    );
    mainPanelLayout.setVerticalGroup(

mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(mainPanelLayout.createSequentialGroup()
        .addContainerGap()

.addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
CENTER)
            .addComponent(Lb_Conect)
            .addComponent(Bt_Conect))
        .addGap(18, 18, 18)

.addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
            .addGroup(mainPanelLayout.createSequentialGroup()

.addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
BASELINE)
                .addComponent(Lb_min)
                .addComponent(Lb_media))

.addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
                .addGroup(mainPanelLayout.createSequentialGroup()
                  .addGap(19, 19, 19)
                  .addComponent(Lb_minV))
                .addGroup(mainPanelLayout.createSequentialGroup()
                  .addGap(18, 18, 18)
                  .addComponent(Lb_mediaV))))
            .addGroup(mainPanelLayout.createSequentialGroup()
```

```
                .addComponent(Lb_max, javax.swing.GroupLayout.PREFERRED_SIZE, 14,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(19, 19, 19)
                .addComponent(Lb_maxV)))
            .addGap(18, 18, 18)
            .addComponent(jSlider1, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(Lb_brilloV)
            .addGap(7, 7, 7)

.addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
CENTER)
              .addComponent(Lb_corrIzq)
              .addComponent(Lb_corrDer))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
66, Short.MAX_VALUE)
            .addComponent(Lb_obj)
            .addGap(18, 18, 18)

.addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
BASELINE)
              .addComponent(jLabel2)
              .addComponent(Lb_distV)
              .addComponent(jLabel1))
            .addGap(11, 11, 11)

.addGroup(mainPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.
LEADING)
              .addComponent(jLabel3)
              .addComponent(Pb_dist, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(48, 48, 48))
        );

        menuBar.setName("menuBar"); // NOI18N

        fileMenu.setText(resourceMap.getString("fileMenu.text")); // NOI18N
        fileMenu.setName("fileMenu"); // NOI18N

        javax.swing.ActionMap actionMap =
org.jdesktop.application.Application.getInstance(pantallappal.PantallaPpalApp.class).g
etContext().getActionMap(PantallaPpalView.class, this);
        exitMenuItem.setAction(actionMap.get("quit")); // NOI18N
        exitMenuItem.setName("exitMenuItem"); // NOI18N
        fileMenu.add(exitMenuItem);
```

```
        menuBar.add(fileMenu);

        helpMenu.setText(resourceMap.getString("helpMenu.text")); // NOI18N
        helpMenu.setName("helpMenu"); // NOI18N

        aboutMenuItem.setAction(actionMap.get("showAboutBox")); // NOI18N
        aboutMenuItem.setName("aboutMenuItem"); // NOI18N
        helpMenu.add(aboutMenuItem);

        menuBar.add(helpMenu);

        statusPanel.setName("statusPanel"); // NOI18N

        statusMessageLabel.setName("statusMessageLabel"); // NOI18N

        statusAnimationLabel.setHorizontalAlignment(javax.swing.SwingConstants.LEFT);
        statusAnimationLabel.setName("statusAnimationLabel"); // NOI18N

        progressBar.setName("progressBar"); // NOI18N

        jLabel4.setFont(resourceMap.getFont("jLabel4.font")); // NOI18N
        jLabel4.setText(resourceMap.getString("jLabel4.text")); // NOI18N
        jLabel4.setName("jLabel4"); // NOI18N

        javax.swing.GroupLayout statusPanelLayout = new
javax.swing.GroupLayout(statusPanel);
        statusPanel.setLayout(statusPanelLayout);
        statusPanelLayout.setHorizontalGroup(

statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(statusPanelLayout.createSequentialGroup()
               .addContainerGap()

.addGroup(statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.LEADING)
                  .addGroup(statusPanelLayout.createSequentialGroup()
                     .addComponent(statusMessageLabel)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 331,
Short.MAX_VALUE)
                     .addComponent(statusAnimationLabel)
                     .addGap(328, 328, 328))
                  .addGroup(statusPanelLayout.createSequentialGroup()
                     .addComponent(progressBar, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                     .addGap(79, 79, 79)
                     .addComponent(jLabel4)
```

```
                .addContainerGap(270, Short.MAX_VALUE))))
        );
        statusPanelLayout.setVerticalGroup(

statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
statusPanelLayout.createSequentialGroup()
                .addContainerGap(22, Short.MAX_VALUE)

.addGroup(statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.BASELINE)
                    .addComponent(statusMessageLabel)
                    .addComponent(statusAnimationLabel))
                .addGap(16, 16, 16)

.addGroup(statusPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignmen
t.CENTER)
                    .addComponent(jLabel4)
                    .addComponent(progressBar,     javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap())
        );

        jFrame1.setName("jFrame1"); // NOI18N

        javax.swing.GroupLayout                jFrame1Layout                =                new
javax.swing.GroupLayout(jFrame1.getContentPane());
        jFrame1.getContentPane().setLayout(jFrame1Layout);
        jFrame1Layout.setHorizontalGroup(

jFrame1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 400, Short.MAX_VALUE)
        );
        jFrame1Layout.setVerticalGroup(

jFrame1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGap(0, 300, Short.MAX_VALUE)
        );

        setComponent(mainPanel);
        setMenuBar(menuBar);
        setStatusBar(statusPanel);
    }// </editor-fold>

    private void Bt_ConectMouseClicked(java.awt.event.MouseEvent evt) {

                conexion = new BTConect();
```

```
                conexion.ConexionBTPC();


                timer.start();
            mainPanel.repaint();


}

private void mainPanelComponentShown(java.awt.event.ComponentEvent evt) {
   // TODO add your handling code here:
}



public void actionPerformed(ActionEvent e)
{
   // Aquí el código que queramos ejecutar.
 }

public void traduccion(String v1){

   int valor=0,i = 0;

   if(v1.startsWith("1"))
   {

      valor=Integer.parseInt(v1.substring(1));
      System.out.println("Maximo= " +  valor);

      jSlider1.setMaximum(valor+100);
      Lb_maxV.setText(Integer.toString(valor));

   }

   if(v1.startsWith("2"))
   {

      valor=Integer.parseInt(v1.substring(1));
      System.out.println("Minimo= " + valor);

      jSlider1.setMinimum(valor);
      Lb_minV.setText(Integer.toString(valor-100));


   }
```

```
if(v1.startsWith("3"))
{

   valor=Integer.parseInt(v1.substring(1));
   System.out.println("Media= " + valor);

   Lb_mediaV.setText(Integer.toString(valor));

}


if(v1.startsWith("4"))
{

   valor=Integer.parseInt(v1.substring(1));
   System.out.println("Distancia= " + valor);

   if(valor>60){

      Pb_dist.setValue(60);

   }else{
      Pb_dist.setValue(valor);
   }

   if (valor<=20){

      Lb_obj.setForeground(Color.red);
      Lb_obj.setText("Obstáculo encontrado");

   }else{

      Lb_obj.setForeground(Color.green);
      Lb_obj.setText("No hay obstáculo");
   }

   Lb_distV.setText(Integer.toString(Pb_dist.getValue()));

}

if(v1.startsWith("5"))
{

   valor=Integer.parseInt(v1.substring(1));
   System.out.println("Brillo= " + valor);
```

```
        if(valor>Integer.parseInt(Lb_maxV.getText())){
            jSlider1.setValue(Integer.parseInt(Lb_maxV.getText()));
        }else{

            if(valor<Integer.parseInt(Lb_minV.getText())){
              jSlider1.setValue(Integer.parseInt(Lb_minV.getText()));
            }else{

                jSlider1.setValue(valor);

            }
        }

        if(jSlider1.getValue()>Integer.parseInt(Lb_mediaV.getText())){

            Lb_corrDer.setForeground(Color.blue);
            Lb_corrIzq.setText("");
            Lb_corrDer.setText("DERECHA");

        }else{

            Lb_corrIzq.setForeground(Color.blue);
            Lb_corrDer.setText("");
            Lb_corrIzq.setText("IZQUIERDA");

        }


        Lb_brilloV.setText(Integer.toString(valor));

    }

    if(v1.startsWith("6")){

        System.exit(0);
    }

}
Timer timer;
BTConect conexion;
// Variables declaration - do not modify
private javax.swing.JButton Bt_Conect;
private javax.swing.JLabel Lb_Conect;
private javax.swing.JLabel Lb_brilloV;
private javax.swing.JLabel Lb_corrDer;
private javax.swing.JLabel Lb_corrIzq;
private javax.swing.JLabel Lb_distV;
```

```
    private javax.swing.JLabel Lb_max;
    private javax.swing.JLabel Lb_maxV;
    private javax.swing.JLabel Lb_media;
    private javax.swing.JLabel Lb_mediaV;
    private javax.swing.JLabel Lb_min;
    private javax.swing.JLabel Lb_minV;
    private javax.swing.JLabel Lb_obj;
    private javax.swing.JProgressBar Pb_dist;
    private javax.swing.JFrame jFrame1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JSlider jSlider1;
    private javax.swing.JPanel mainPanel;
    private javax.swing.JMenuBar menuBar;
    private javax.swing.JProgressBar progressBar;
    private javax.swing.JLabel statusAnimationLabel;
    private javax.swing.JLabel statusMessageLabel;
    private javax.swing.JPanel statusPanel;
    // End of variables declaration

    private final Timer messageTimer;
    private final Timer busyIconTimer;
    private final Icon idleIcon;
    private final Icon[] busyIcons = new Icon[15];
    private int busyIconIndex = 0;
    private JDialog aboutBox;
}
```

A continuación se presenta un enlace a un video, en el que se puede ver esta versión funcionando junto a la aplicación para PC.

http://www.youtube.com/watch?v=Ha4Ym0SJ2aE

4. Tercer programa(Threads)

Para el tercer programa se pretende que el robot siga una línea, al igual que el objetivo marcado para los anteriores programas, pero incorporando una manera diferente de afrontar el desafio con el uso de hilos y la capacidad de java para el Multithreading.

4.1 Introducción.

Como se dice en las líneas anteriores, el objetivo del programa vuelve a ser que el robot siga una línea, pero con un cambio importante en el planteamiento de cómo atacar ese objetivo.

4.2 Objetivo

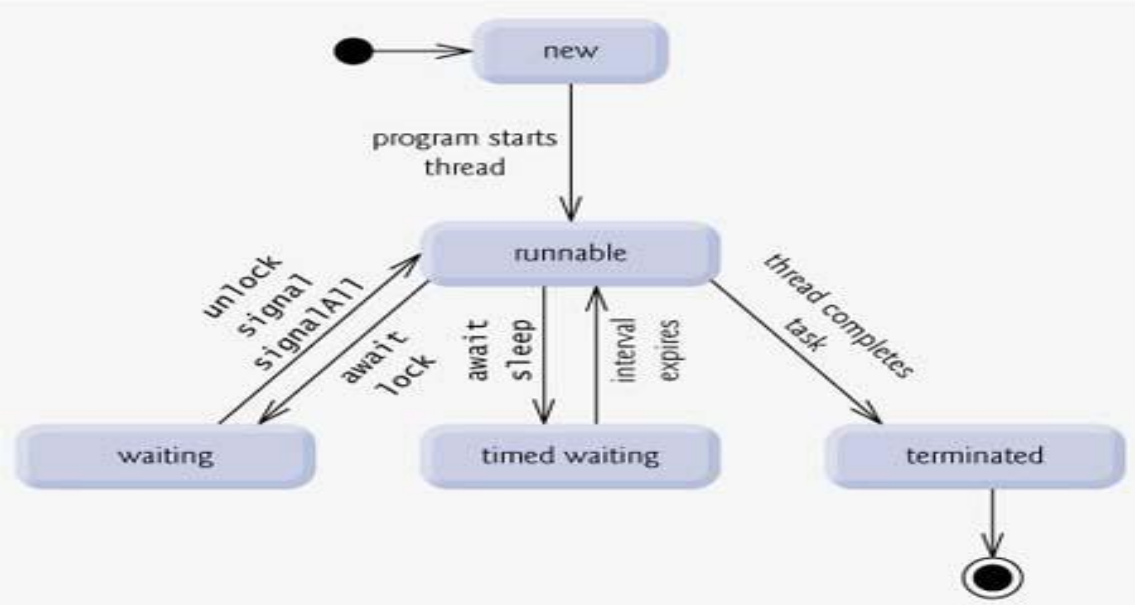
Se pretende alcanzar el objetivo marcado empleando las clases que se crearon para el segundo programa, con algunas pequeñas modificaciones que se detallarán más adelante, y mediante el uso de hilos.

Los *threads* o hilos de ejecución permiten organizar los recursos del ordenador de forma que pueda haber varios programas actuando en paralelo. Un hilo de ejecución puede realizar cualquier tarea que pueda realizar un programa normal y corriente.

Un *thread* o hilo es un flujo secuencial simple dentro de un proceso, un único proceso puede tener varios hilos ejecutándose. Java es el único lenguaje de uso general en el que la compatibilidad con los hilos forma parte del lenguaje.

Ciclo de vida de un hilo

El hilo pasa por diferentes etapas de su ciclo de vida. Por ejemplo, un hilo que nace, se inicia, funciona, y luego muere. El siguiente diagrama muestra el ciclo de vida completo de un hilo.



Las etapas antes mencionadas se explican a continuación:

• Nuevo: un nuevo hilo comienza su ciclo de vida en el nuevo estado. Permanece en este estado hasta que el programa inicia el hilo.

• Ejecutable: Después de un hilo recién nacido se inicia, el hilo se vuelve ejecutable. Un hilo en este estado es considerado como la ejecución de sus tareas.

• En espera: A veces un hilo de transición al estado de espera mientras el subproceso espera a que otro hilo, para realizar una transición o tarea. El estado ejecutable sólo cuando otro hilo señala que el hilo está a la espera de continuar con la ejecución.

• Tiempo de espera en espera: un subproceso ejecutable puede entrar en el estado de tiempo de espera para un intervalo de tiempo especificado. Un hilo de este estado de transición de vuelta al estado ejecutable en ese intervalo de tiempo de vencimiento o cuando el evento se espera ocurra.

• Terminado: Una hebra ejecutable entra en el estado terminado cuando termina su tarea, o de lo contrario termina.

Prioridades de los subprocesos:

Cada hilo de Java tiene una prioridad que ayuda al sistema operativo determinar el orden en que las discusiones se han programado.

Las prioridades de Java están en el rango entre MIN_PRIORITY (una constante de 1) y MAX_PRIORITY (una constante de 10). Por defecto, cada enlace se le da prioridad NORM_PRIORITY (una constante de 5).

Temas de mayor prioridad son más importantes para un programa y debe ser asignado antes de tiempo de procesador de menor prioridad discusiones. Sin embargo, las prioridades de hilo no puede garantizar el orden, dependen de la plataforma mucho.

Creación de un mensaje:

Java define dos formas en que esto se puede lograr:

• Se puede implementar la interfaz ejecutable.

• Se puede extender la clase Hilo, sí. Crear un mensaje privado mediante la implementación de ejecutable:

La forma más fácil de crear un hilo es crear una clase que implementa la interfaz ejecutable.

Para implementar ejecutable, una clase solo necesita implementar un solo método llamado ejecutar ().

Va a definir el código que constituye el hilo nuevo dentro de un método ejecutar (). Es importante entender que el ejecutar () puede llamar a otros métodos, el uso de otras clases, y declarar las variables, puede al igual que el hilo principal.

Después de crear una clase que implementa el ejecutable, se crea un objeto de tipo hilo dentro de esa clase. Define varios constructores. El que vamos a utilizar se muestra aquí:

ThreadOb es una instancia de una clase que implementa la interfaz ejecutable y el nombre del nuevo hilo se especifica threadName.

Se crea un nuevo hilo , que no comenzará a correr hasta que llame a su método start (), que se declara dentro de un mensaje privado. El método start ().

A continuación se presenta la lista de medthods importantes disponibles en la clase Thread.

SN	Methods with Description
1	**public void start()** Starts the thread in a separate path of execution, then invokes the run() method on this Thread object.
2	**public void run()** If this Thread object was instantiated using a separate Runnable target, the run() method is invoked on that Runnable object.
3	**public final void setName(String name)** Changes the name of the Thread object. There is also a getName() method for retrieving the name.
4	**public final void setPriority(int priority)** Sets the priority of this Thread object. The possible values are between 1 and 10.
5	**public final void setDaemon(boolean on)** A parameter of true denotes this Thread as a daemon thread.
6	**public final void join(long millisec)** The current thread invokes this method on a second thread, causing the current thread to block until the second thread terminates or the specified number of milliseconds passes.
7	**public void interrupt()** Interrupts this thread, causing it to continue execution if it was blocked for any reason.
8	**public final boolean isAlive()** Returns true if the thread is alive, which is any time after the thread has been started but before it runs to completion.

Los métodos anteriores se invocan en un objeto Thread particular. Los siguientes métodos de la clase Thread son estáticos. La invocación de uno de los métodos estáticos realiza la operación en el hilo en ejecución.

SN	Methods with Description
1	**public static void yield()** Causes the currently running thread to yield to any other threads of the same priority that are waiting to be scheduled
2	**public static void sleep(long millisec)** Causes the currently running thread to block for at least the specified number of milliseconds
3	**public static boolean holdsLock(Object x)** Returns true if the current thread holds the lock on the given Object.
4	**public static Thread currentThread()** Returns a reference to the currently running thread, which is the thread that invokes this method.
5	**public static void dumpStack()** Prints the stack trace for the currently running thread, which is useful when debugging a multithreaded application.

4.3 Desarrollo

Conviene identificar claramente cuáles son los puntos más importantes de este nuevo objetivo y en qué va a afectar al programa anterior. Por un lado se convierten todos los procesos en hilos, es decir los comportamientos también se convierten en hilos.

Las condiciones para el movimiento consisten en que no haya ningún objeto delante del robot, mientras que la condición de parada es la opuesta. En cuanto a la parte relacionada con el movimiento del robot, se intenta que el robot siempre este encima de la línea negra, para ello se realiza una calibración previa.

4.4 Clases definidas

4.4.1 Clase Contenedor

La clase Contenedor es la encargada de albergar las variables que se vayan a emplear a lo largo del código. De esta manera, si algún objeto necesita conocer el valor de brillo o potencia de los motores por ejemplo, deberá acudir a la variable determinada de la clase Contenedor y leerla.

El código de la clase Contenedor es:

```
public class Contenedor {
	private int powerB, powerC, modeB, modeC, power;
	private int max,min,media,cmd, distancia;
	private int brillo;

	Contenedor(){
		distancia = 0;
		brillo = 0;
		power = 0;
		powerB = 0;
		powerC = 0;
		modeB = 0;
		modeC = 0;
		max = 0;
		min = 0;
		media = 0;
		cmd = 1;
	}

	public void setDist(int v1){
		distancia = v1;
	}

	public int getDist(){
		return distancia;
	}

	public void setBrillo(int v1){
		brillo = v1;
	}

	public int getBrillo(){
		return brillo;
	}

	public void setPowerB(int v1){
		 powerB = v1;
	 }
```

```
public int getPowerB(){
	return powerB;
}

 public void setPowerC(int v1){
	 powerC = v1;
 }

 public int getPowerC(){
		return powerC;
	}

 public void setModeB(int v1){
	 modeB = v1;
 }

public int getModeB(){
	return modeB;
}

 public void setModeC(int v1){
	 modeC = v1;
 }

public int getModeC(){
	return modeC;
}

public void setPower(int v1){
	power = v1;
}

public int getPower(){
	return power;
}

public void setMax(int v1){
	max = v1;
}

public int getMax(){
	return max;
}

public void setMin(int v1){
	min = v1;
}

public int getMin(){
	return min;
}

public void setMedia(){
	media = (getMax() + getMin()) / 2;
}

public int getMedia(){
	return media;
}
```

```
public void setCMD(int comando){
	cmd = comando;
}

public int getCMD(){
	return cmd;
}
	}
```

4.4.2 Clase Motores

En esta clase jugamos con la potencia de los motores. Cuando más potencia más velocidad, con menos potencia menos velocidad. Parámetros que utilizamos son:

3. Power ; que tiene que tener un valor de 0 a 100.

4. Modo; se define en base Puerto del motor.

   a. 1 = hacia delante
   b. 2 = hacia atrás
   c. 3 = paro
   d. 4 = float

Este método hace que el robot se desplace hacia delante.

```
public void adelante(int powerB,int powerC)
		{
		 MotorPort.B.controlMotor(powerB, 1);
		 MotorPort.C.controlMotor(powerC, 1);
		}
```

Este método hace que el robot se desplace atrás.

```
public void atras(int powerB,int powerC)
	{
	 MotorPort.B.controlMotor(powerB, 2);
	 MotorPort.C.controlMotor(powerC, 2);
	}
```

Este método hace que el robot se gire a la derecha. Para realizar el giro para el motor de la izq.

```
public void girarder(int powerB,int powerC)
      {
       MotorPort.B.controlMotor(powerB, 1);
       MotorPort.C.controlMotor(powerC, 3);
      }
```

Este método hace que el robot se gire a la izquierda. Para realizar el giro para el motor de la derecha.

```
public void girarizq(int powerB,int powerC)
      {
       MotorPort.B.controlMotor(powerB, 3);
       MotorPort.C.controlMotor(powerC, 1);
      }
```

Este método hace que el robot se gire a la izquierda, derecha, delante y atrás.

```
public void calibrado(int powerB,int powerC,int modeB,int modeC)
      {
       MotorPort.B.controlMotor(powerB, modeB);
       MotorPort.C.controlMotor(powerC, modeC);
      }
```

Este método actualiza los valores de la clase contenedor.

```
public void setpowerB(int powerB)
      {
       contenedor.powerB=powerB;

      }

public void setpowerC(int powerC)
      {
       contenedor.powerC=powerC;
      }

 public void setmodeB(int modeB)
      {
       contenedor.modeB=modeB;
      }

 public void setmodeC(int modeC)
      {
```

```
contenedor.modeC=modeC;
}
```

4.4.3 Clase Sensores

Esta clase contiene todos los métodos que usamos para controlar ambos sensores:

```
import lejos.nxt.ColorSensor;
import lejos.nxt.SensorPort;
import lejos.nxt.UltrasonicSensor;


public class Sensores {
	Contenedor cont;
	private UltrasonicSensor sus;
	private ColorSensor sc;

		/*
		 * Metodo constructor de la clase.
		*/
		public Sensores(Contenedor v1){
			cont = v1;
			sc = new ColorSensor(SensorPort.S3);
			sus = new UltrasonicSensor(SensorPort.S4);
		}

		/*
		 * Metodo que actualize el valor de distancia en la
		 * clase Contenedor.
		*/
		public void getDistancia(){
			cont.setDist(sus.getDistance());
		}

		/*
		 * Metodo que devuelve true si la distancia es menor o
		 * igual a la indicada y false en caso contrario.
		*/
		public boolean isCerca(int v1){

					boolean rdo = false;

			if (cont.getDist() <= v1){
				rdo = true;
			}
			else{
				rdo = false;
			}
			return rdo;
		}
```

```
        /*
         * Metodo que actualize el valor del brillo en la
         * clase Contenedor.
        */
        public void Brillo(){
            cont.setBrillo(sc.getNormalizedLightValue());
        }

        /*
         * Metodo que actualize el valor máximo del brillo en la
         * clase Contenedor.
        */
        public void BrilloMax(){
            cont.setMax(sc.getNormalizedLightValue());
        }

        /*
         * Metodo que actualize el valor minimo del brillo en la
         * clase Contenedor.
        */
        public void BrilloMin(){
            cont.setMin(sc.getNormalizedLightValue());
        }

        /*
         * Metodo que inicia el sensor de color como lampara
         * clase Contenedor.
        */
        public void Luz(boolean v1){
            sc.setFloodlight(v1);
        }
}
```

4.4.4 Clase BTnxt

Esta clase se encarga de conectar por bluetooth con el PC y si no logra sigue la ejecución normal del programa, además del envio de datos cuando se ha producido la conexión:

```
import java.io.DataOutputStream;
import java.io.IOException;
import lejos.nxt.LCD;
import lejos.nxt.comm.BTConnection;
import lejos.nxt.comm.Bluetooth;


public class BTnxt {
	BTConnection conex;
	DataOutputStream salida;

	public BTnxt(){
```

```
		try {
			conex = Bluetooth.waitForConnection(5000, 0);
			if (conex != null){
				salida = conex.openDataOutputStream();
			}
		} catch (Exception e){}
	}


	 /*
	 * Este metodo devuelve un booleano, true si la conexion esta hecha
	 * y false si nop ha conectado por bluetooth.
	 */
	 public boolean getEstado() {
		boolean rdo = false;
	 if(conex != null ){
		rdo=true;
	 }
	 else {
		rdo=false;
	 }
	 return rdo;
	 }


	/*
	 * Este metodo recoge 1 string y 1 entero, los convierte en un string
para enviarlos al
	 * PC por bluetooth.
	 */
	void enviarmsg(String msg, int valor){
	  String mensaje;
	  mensaje = msg + String.valueOf(valor);
	  try {
			salida.writeUTF(mensaje);
			salida.flush();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}


	/*
	 * Se cierra el canal de salida de datos
	 * ademas de la conexion con el PC.
	 */
	void cerrar(){
		try {
			salida.close();
			Thread.sleep(100);
		}
		catch (IOException e1) {e1.printStackTrace();}
		catch (InterruptedException e) {e.printStackTrace();}
		conex.close();
		LCD.clear();
	}
}
```

4.5 Clases threads o hilos.

4.5.1 Clase HiloCorregir

Es este hilo el responsable de ir siguiendo la línea negra comparando los valores que lee con el sensor de color y comparándolo con el valor guardado en la clase contenedor:

```
public class HiloCorregir extends Thread{
	private Contenedor cont;
	private Sensores sensor;
	private Motores piloto;

	public HiloCorregir(Contenedor v1, Sensores v2){
		cont = v1;
		piloto = new Motores(cont);
		sensor = v2;
	}

	public void run(){
		while(true)
		{
			if(cont.getCMD() ==1){
				sensor.Brillo();
				if(cont.getBrillo() < cont.getMedia()){
					piloto.girarder(90);
				}
				else {
					piloto.girarizq(90);
				}
			}
		}
	}
}
```

4.5.2 Clase HiloCorregir

Este hilo se encarga de que el robot no choque contra ningún obstáculo mientras sigue la línea negra y se quedara parado pitando hasta que se retire el obstáculo.

```
public class HiloDetectar extends Thread{
	private Contenedor cont;
	Sensores sensor;
	private final int distancia =15;

	public HiloDetectar(Contenedor v1){
```

```
		cont = v1;
		sensor = new Sensores(cont);
	}

	public void run(){
		while(true)
		{
			sensor.getDistancia();
			if(cont.getDist() > distancia){
				cont.setCMD(1);
			}
			else {
				cont.setCMD(0);
			}
		}
	}
}
```

4.5.3 Clase HiloEnviar

Este hilo se encarga de mandar al PC (siempre y cuando se haya establecido una conexión por bluetooth) los datos guardados en las variables de la clase Contenedor.

```
public class HiloEnviar extends Thread{
	private Contenedor cont;
	private BTnxt bt;

	public HiloEnviar(Contenedor v1, BTnxt v2){
		cont = v1;
		bt = v2;
	}

	public void run(){
		while(true){
			bt.enviarmsg("SensorColor: ", cont.getBrillo());
			bt.enviarmsg("SensorUS: ", cont.getDist());
			bt.enviarmsg("Power: ", cont.getPower());
			bt.enviarmsg("Media: ", cont.getMedia());
			bt.enviarmsg("Maximo: ", cont.getMax());
			bt.enviarmsg("Minimo: ", cont.getMin());
		}
	}
}
```

4.5.4 Clase HiloParar

Este hilo se encarga de detener el robot mientras el obstáculo esta delante de el:

```
import lejos.nxt.Sound;

public class HiloParar extends Thread{
	private Contenedor cont;
	private Motores piloto;

	public HiloParar(Contenedor v1){
		cont = v1;
		piloto = new Motores(cont);
	}

	public void run(){
		while(true){
			if(cont.getCMD() == 0){
				piloto.parar();
				Sound.twoBeeps();
			}
		}
	}
}
```

4.6 Programa Principal.

El programa principal será el espacio en el que se escribirá el código que crea los objetos que se van a emplear.

El código correspondiente es el siguiente:

```
import lejos.nxt.*;

public class Main {
	public static void main(String[] args){
		Contenedor cont;
		Sensores sensor;
		HiloEnviar enviar;
		HiloParar parado;
		HiloCorregir corrige;
		HiloDetectar detecta;
```

```
		boolean conexOn;

		BTnxt bt = new BTnxt();
		cont = new Contenedor();
		sensor = new Sensores(cont);
		conexOn = false;
		conexOn = bt.getEstado();

		// Se calcula el valor máximo de luz
		while(!Button.LEFT.isPressed()){
			sensor.BrilloMax();
			LCD.drawInt(cont.getMax(), 0, 0);
		}

		// se calcula el valor minimo de la luz(cinta negra).
		while(!Button.RIGHT.isPressed()){
			sensor.BrilloMin();
			LCD.drawInt(cont.getMin(), 1, 1);
			cont.setMedia();
			LCD.drawInt(cont.getMedia(), 2, 2);
		}

		parado = new HiloParar(cont);
		corrige = new HiloCorregir(cont, sensor);
		detecta = new HiloDetectar(cont);

		if(conexOn){
			enviar = new HiloEnviar(cont, bt);
			enviar.start();
		}
		parado.start();
		detecta.start();
		corrige.start();

		// Condicion de parada
		while(!Button.ESCAPE.isPressed()){
		}
		if(conexOn){
			bt.cerrar();
		}
		System.exit(0);
	}
}
```

4.7 Programa PC.

4.7.1 Clase Metodos

Esta clase se encarga de conectar por bluetooth con el NXT y de la recepción de datos cuando se ha producido la conexión:

```
import java.io.DataInputStream;
import java.io.IOException;
```

```
import lejos.pc.comm.*;

public class Metodos {
        private NXTConnector conex;
        private DataInputStream entrada;
        private String mensajeNXT;
        private boolean conectado;
        private String nombre;
        private String mac;

        public Metodos(){
              conex = new NXTConnector();
              nombre = "NXT";
              mac = "00165311E835";
              conectado = false;
        }

        void ConexionBTPC(){
              System.out.println(" Conectandose al " + nombre + mac);
              conectado = conex.connectTo(nombre, mac,
NXTCommFactory.BLUETOOTH);
              if (!conectado) {
                        System.err.println("Fallo al conectar al NXT");
                        System.exit(1);
              }
              else {
                    entrada = conex.getDataIn();
              }
        }

        public boolean getEstado(){
              return conectado;
        }

        void recibir(){
            try
            {
              mensajeNXT = entrada.readUTF();
              System.out.println(mensajeNXT);
            } catch (IOException e) {System.out.println("Error de lectura" +
e);}
        }

        void CerrarConex(){
              try {
                    entrada.close();
                    conex.close();
              } catch (IOException e) {e.printStackTrace();}
        }
      }
```

4.7.2 Programa Principal

El programa principal será el espacio en el que se escribirá el código que crea los objetos que se van a emplear.

```
public class Main {

	public static void main(String[] args) {
		Metodos a = new Metodos();
		a.ConexionBTPC();
		while (a.getEstado()){
			a.recibir();
		}
		a.CerrarConex();

	}
}
```

El enlace que se muestra a continuación, es un video en el que se puede ver el robot siguiendo un circuito en el que se incluyen curvas a izquierda y derecha, y una recta. Además se incluye una vista subjetiva desde el propio NXT.

http://www.youtube.com/watch?v=ZSelVKkFMus