CIM-NXT

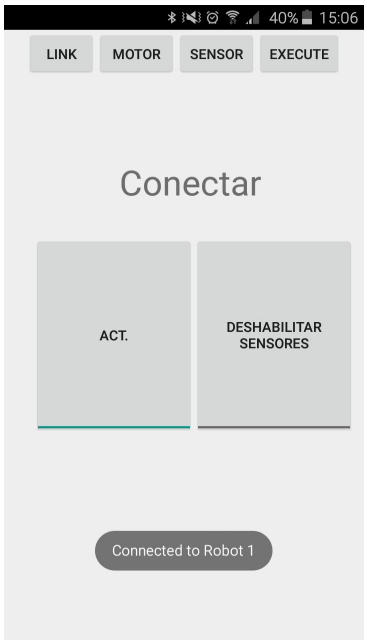
Inicio de la aplicación (menú Link):

Dentro de la aplicación presione el botón “DESACTIVADO” para activar el bluetooth del teléfono, luego se desplegará una lista de dispositivos a los que se puede conectar la aplicación vía bluetooth.

Nota: la primera vez que se abra la aplicación, esta no reconocerá directamente al robot, Por lo cual, usted debe presionar el botón “scan” y se actualizará la lista de dispositivos tipo NXT. (si no aparece ningún dispositivo, revise que el robot NXT se encuentre encendido y tenga el bluetooth activado). La primera conexión pedirá un número de pin en el robot y en el Smartphone que por defecto es 1234 (Los dispositivos deben estar reconocidos para continuar).

Una vez se actualice la lista, presione el que corresponde al robot que desea controlar. Alrededor de 1 a 5 segundos el robot debe sonar (si el sonido desde el robot está habilitado), y se dara aviso de que el robot está conectado con su teléfono por medio de un mensaje desplegado en la pantalla del smartphone

.



Una vez conectado el robot NXT con un dispositivo móvil a través de la aplicación se podrá acceder a todos los contenidos de esta los cuales serán explicados a continuación siendo separados por su funcionalidad a través de menús.

Menús de opciones

- Menú “Motor”

U:mover el robot hacia adelante.
D:mover el robot hacia atras.
R: mover el robot hacia la derecha.
L:mover el robot hacia la izquierda.
GL:girar el robot sobre su eje hacia la izquierda.
GR:girar el robot sobre su eje hacia la derecha.
Potencia de motores:barra que permite modificar manualmente la potencia con la que trabajan los motores del robot.
U3: mover el brazo del robot hacia arriba.
D3:mover el brazo del robot hacia abajo.

Nota:todos los botones de este menú se ejecutarán el tiempo que estos sean presionados por el usuario.

- Menú “Execute”

Reacción(10 cm): El robot anda hacia adelante hasta que encuentra un obstáculo a 10 cm y gira una rueda 2 vueltas y luego hace 3 rotaciones hacia adelante.

Reacción(15 cm): El robot anda hacia adelante hasta que encuentra un obstáculo a 15 cm y hace 3 rotaciones hacia atrás y luego gira una rueda en 2 vueltas.

Reacción(20 cm): El robot anda hacia adelante hasta que encuentra un obstáculo a 20 cm y gira una rueda 2 vueltas y luego hace 3 rotaciones hacia atrás.

Reacción(25 cm): El robot anda hacia adelante hasta que encuentra un obstáculo a 25 cm y hace 3 rotaciones hacia adelante y luego gira una rueda en 2 vueltas.

Spinner de Dirección y Cantidad:Permite seleccionar de una lista de valores fijos la dirección y cantidad que se interpretara como rotación o tiempo.

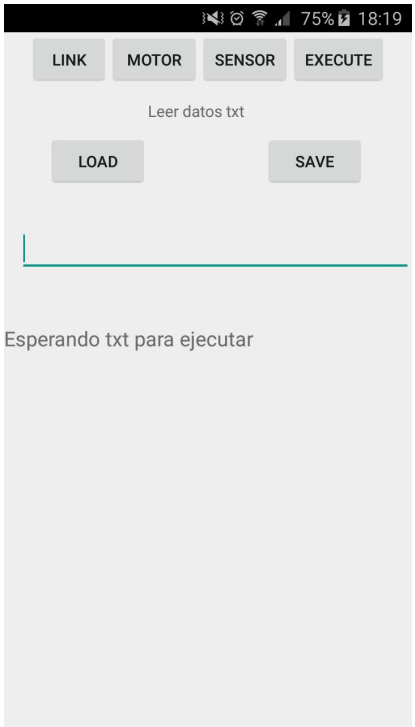
Rotación:Ejecuta los parámetros de cantidad como número de rotaciones hacia la direccion seleccionada.

Tiempo:Ejecuta los parámetros de cantidad como tiempo (segundos) de ejecución hacia la direccion seleccionada.

Leer Datos: se desplegará una nueva ventana que permitirá al usuario manipular el robot a través de un archivo .txt por medio de la aplicación CIM-NXT u otra aplicación de editor de texto o bien a través de un .txt almacenado en la memoria interna del celular enviado desde un computador u otro equipo.

Servidor: se despliega una nueva ventana que permite la conexión por medio de wifi de un cliente que envía instrucciones al servidor para ser ejecutados por el robot.

- **Leer Datos:**



SAVE: crea una carpeta en memoria interna llamada "DatosNXT" (si es que no ha sido creada) donde se almacenará un archivo txt con nombre "setnxt.txt" con un conjunto de instrucciones que se podrán escribir desde la aplicación a través de teclado desde el panel de escritura.

LOAD:carga el archivo "setnxt.txt" desde la carpeta en memoria interna llamada "DatosNXT" y la ejecución de las instrucciones almacenadas en ellas al robot NXT.

"Esperando txt para ejecutar": panel donde se muestran las instrucciones cargadas desde el archivo setnxt.txt.

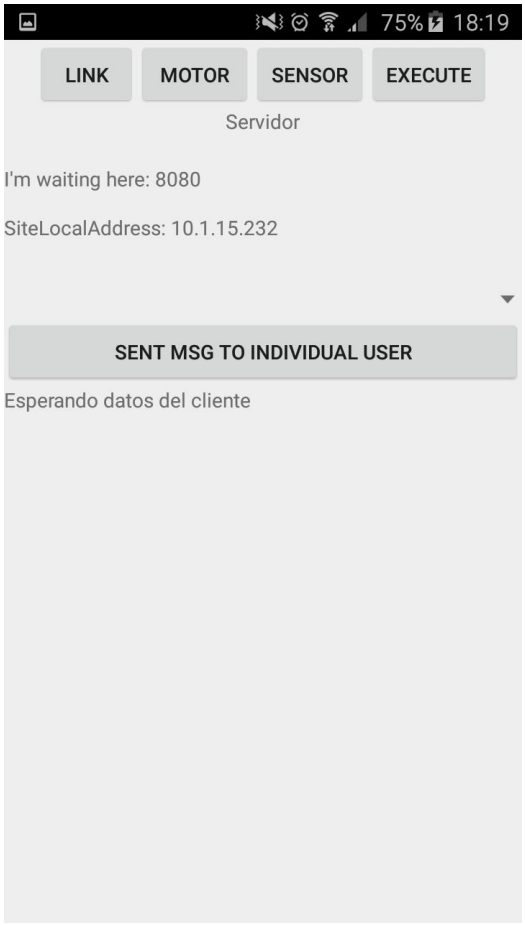
Instrucciones permitidas en el documento setnxt.txt:

w:mover el robot hacia adelante.
s:mover el robot hacia atras.
d: mover el robot hacia la derecha.
a:mover el robot hacia la izquierda.
ga:girar el robot sobre su eje hacia la izquierda.
gd:girar el robot sobre su eje hacia la derecha.
bw: mover el brazo del robot hacia arriba.
bs:mover el brazo del robot hacia abajo.
fin:terminar la ejecución.
1-10:números permitidos para el tiempo de cada instrucción.

Nota: el archivo "setnxt.txt"permite un formato de **dirección** salto de línea **tiempo de ejecución** que se realizarán hasta que el documento finalice con un **fin**, todo debe estar escrito con letras minúsculas.

Ejemplo de instrucciones válidas:
w
5
s
3
fin

- **Servidor**



SiteLocaAddress:x.x.x.x :muestra la dirección ip del servidor (x.x.x.x) la cual debe ingresar manualmente el cliente para poder enviar instrucciones al robot NXT a través de señal wifi .

SENT MDG TO INDIVIDUAL USER: permite al servidor enviar un mensaje a un cliente conectado para verificar la conexión entre dispositivos , si la conexión está correcta el mensaje se desplegará por la pantalla del cliente.

“Esperando datos del cliente”: panel donde se muestran los mensajes que envia el cliente al servidor.

Nota: los mensajes se ejecutarán en la aplicación desde Servidor con los mensajes que el cliente envía.

Mensajes
1:mover el robot hacia adelante.
2:mover el robot hacia atras.
3:girar el robot sobre su eje hacia la izquierda.
4:girar el robot sobre su eje hacia la derecha.
5: mover el brazo del robot hacia arriba.
6:mover el brazo del robot hacia abajo.

Funciones Claves del Código

```
public void motors3(byte l, byte r, byte action, boolean speedReg, boolean motorSync) {
    byte[] data = {0x0c, 0x00, (byte) 0x80, 0x04, 0x02, 0x32, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
            0x0c, 0x00, (byte) 0x80, 0x04, 0x01, 0x32, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
            0x0c, 0x00, (byte) 0x80, 0x04, 0x00, 0x32, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
    };
    data[5] = l;
    data[19] = r;
    data[33] = action;
    if (speedReg) {
        data[7] |= 0x01;
        data[21] |= 0x01;
    }
    if (motorSync) {
        data[7] |= 0x02;
        data[21] |= 0x02;
    }
    write(data);
}
```

- Esta función se utiliza mayormente en los botones del menú motor, los movimientos del robot. La función recibe el “byte” que corresponde al motor que se utiliza, data[5] y data[19] corresponden a los motores de las ruedas del robot y data[33] corresponde al motor del brazo del robo y recibe la potencia de los motores que se entrega por medio de la barra de potencia motores.

- Lo que se encuentra en el arreglo byte [] data= { } cada línea representa a un motor del robot, los valores fueron extraídos de otra aplicación de código libre que se encuentra dentro de los archivos de la documentación.

```
//Movimientos manuales del robot
Arriba.setOnTouchListener((v, event) → {
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                moverMotores((byte) 1, (byte) 1, (byte) 0);
                return true;
            case MotionEvent.ACTION_UP:
                moverMotores((byte) 0, (byte) 0, (byte) 0);
                return true;
        }
        return false;
});

Abajo.setOnTouchListener((v, event) → {
        switch (event.getAction()) {
            case MotionEvent.ACTION_DOWN:
                moverMotores((byte) -1, (byte) -1, (byte) 0);
                return true;
            case MotionEvent.ACTION_UP:
                moverMotores((byte) 0, (byte) 0, (byte) 0);
                return true;
        }
        return false;
});
```

- Esta es la estructura de las funciones que se utilizan para manejar manualmente el robot, en este caso solo presentamos los movimientos arriba y abajo, se crea un switch que permite saber si se está presionando o no el botón, en el ejemplo “u” o “d”, del panel motores, si se está presionando se llama a la función moverMotores con los valores correspondientes y si se deja de presionar los valores cambian a 0.

- Para el movimiento arriba, se pasan los valores 1 a los byte que corresponden a los motores como se explicó anteriormente, el valor 1 significa que el robot va a ejecutar la instrucción con potencia= potencia*1, por lo cual en el movimiento abajo el valor es -1, por lo que la potencia de ejecución sería potencia= potencia*(-1) lo que produce un movimiento “negativo” por parte del robot, el robot vuelve atrás.

```
public synchronized void rotacion_arriba(int x) {
    byte[] data = {0x0c, 0x00, (byte) 0x80, 0x04, 0x02, 0x32, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
            0x0c, 0x00, (byte) 0x80, 0x04, 0x01, 0x32, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
    };
    x=x*2;
    data[19] = (byte) 30;
    data[5] = (byte) 30;
    data[10] = (byte) 180;
    data[24] = (byte) 180;
    for (int i = 1; i <= x; i++) {
        //por cada  write(data) solo hace rotacion de 180°
        write(data);
    }
}

public synchronized void rotacion_abajo(int x) {
    byte[] data = {0x0c, 0x00, (byte) 0x80, 0x04, 0x02, 0x32, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
            0x0c, 0x00, (byte) 0x80, 0x04, 0x01, 0x32, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
    };
    x=x*2;
    data[19] = (byte) -30;
    data[5] = (byte) -30;
    data[10] = (byte) 180;
    data[24] = (byte) 180;
    for (int i = 1; i <= x; i++) {
        //por cada  write(data) solo hace rotacion de 180°
        write(data);
    }
}
```

- En este caso los valores que corresponden a la potencia son estáticos, esta función recibe la cantidad de rotaciones por parámetro, pero la rotación de la rueda es de 180°, por lo que se debe multiplicar la cantidad de rotaciones por 2 para que se cumpla la instrucción establecida.

```
public void girar_derecha(int x) {
    byte[] data = {0x0c, 0x00, (byte) 0x80, 0x04, 0x02, 0x32, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
            0x0c, 0x00, (byte) 0x80, 0x04, 0x01, 0x32, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
            //0x0c, 0x00, (byte) 0x80, 0x04, 0x00, 0x32, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
    };
    x=x*2;
    data[19] = (byte) 30;
    data[5] = (byte)  -30;
    data[10] = (byte) 180;
    data[24] = (byte) 180;
    for (int i = 1; i <= x; i++) {
        //por cada  write(data) solo hace rotacion de 180°
        write(data);
    }
}

public void girar_izquierda(int x) {
    byte[] data = {0x0c, 0x00, (byte) 0x80, 0x04, 0x02, 0x32, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
            0x0c, 0x00, (byte) 0x80, 0x04, 0x01, 0x32, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
            //0x0c, 0x00, (byte) 0x80, 0x04, 0x00, 0x32, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
    };
    x=x*2;
    data[19] = (byte) -30;
    data[5] = (byte) 30;
    data[10] = (byte) 180;
    data[24] = (byte) 180;
    for (int i = 1; i <= x; i++) {
        //por cada  write(data) solo hace rotacion de 180°
        write(data);
    }
}
```

- En estas funciones para que el robot gire en su propio eje, en este caso por rotaciones, los motores giran en direcciones contrarias, por eso un data tiene un valor positivo y el otro es negativo.

```
public void tiempo_derecha_empezar() {
    byte[] data = {0x0c, 0x00, (byte) 0x80, 0x04, 0x02, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
            0x0c, 0x00, (byte) 0x80, 0x04, 0x01, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
            //0x0c, 0x00, (byte) 0x80, 0x04, 0x00, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
    };
    data[5] = (byte) 30;
    data[19] = (byte) 40;
    write(data);
}

public void tiempo_izquierda_empezar() {
    byte[] data = {0x0c, 0x00, (byte) 0x80, 0x04, 0x02, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
            0x0c, 0x00, (byte) 0x80, 0x04, 0x01, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
            //0x0c, 0x00, (byte) 0x80, 0x04, 0x00, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00, 0x00, 0x00,
    };
    data[5] = (byte) 40;
    data[19] = (byte) 30;
    write(data);
}
```

- Para resolver el problema del movimiento del robot en curva, un motor debe trabajar con potencia mayor al otro motor, con esto una rueda del motor va a ir más rápido que la otra por lo que genera el movimiento en curva.

```
//Evento del botón para que el robot avance por tiempo en la dirección y cantidad que el usuario seleciono
Tiempo.setOnClickListener((v) → {
                                        if(opcion2==1){
                                            mNXTTalker.tiempo_adelante_empezar();
                                            try {
                                                Thread.sleep(opcion*1000);
                                            } catch (InterruptedException e) {
                                                e.printStackTrace();
                                            }
                                            mNXTTalker.tiempo_parar();}
```

- Para que las instrucciones del robot se ejecuten en un tiempo establecido, se crea un thread.sleep por la cantidad de tiempo, en este caso se trabaja con N segundos (1 segundo =1000), opción es una variable de tipo entero que se usa para dar la cantidad de tiempo, por ejemplo: si se quiere que el robot ejecute una instrucción por 5 segundos el thread debe tener 5000 (5*1000) como parámetro.

```
for (int i = 0; i < loadText.length; i=i+2) {
    C=loadText[i];
    T=loadText[i+1];
    //Verificar la cantidad de tiempo(segundos) que se ejecutara la instrucción dada
    if((resultado=T.startsWith("1")) == true ){r=1;}
    if((resultado=T.startsWith("2")) == true ){r=2;}
    if((resultado=T.startsWith("3")) == true ){r=3;}
    if((resultado=T.startsWith("4")) == true ){r=4;}
    if((resultado=T.startsWith("5")) == true ){r=5;}
    if((resultado=T.startsWith("6")) == true ){r=6;}
    if((resultado=T.startsWith("7")) == true ){r=7;}
    if((resultado=T.startsWith("8")) == true ){r=8;}
    if((resultado=T.startsWith("9")) == true ){r=9;}
    if((resultado=T.startsWith("10")) == true ){r=10;}
    //Verificar la dirección que se ejecutara desde el txt
    if((resultado=C.startsWith("w")) == true ){mNXTTalker.tiempo_adelante_empezar();
        try {
            Thread.sleep(r*1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        mNXTTalker.tiempo_parar();
    }
    if((resultado=C.startsWith("s")) == true ){mNXTTalker.tiempo_atras_empezar();
        try {
            Thread.sleep(r*1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        mNXTTalker.tiempo_parar();
    }
```

-Parte de funcion Load, se utiliza startsWith(“X”) para saber cuánto tiempo se ejecutara la instrucción almacenada en el String T, el cual contiene las instrucciones tanto de movimiento como de tiempo del documento de texto setnxt.txt.