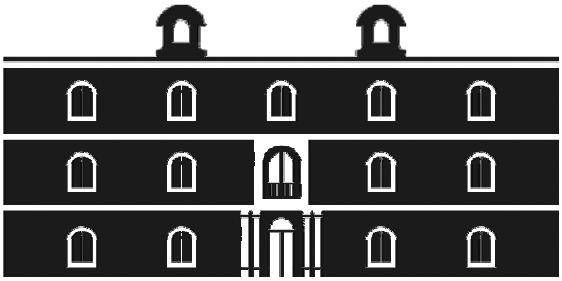
Universidad Politécnica de Cartagena



industriales

etsii UPCT

Escuela Técnica Superior de Ingeniería Industrial

Maqueta de Lego para el seguimiento de un objetivo móvil. Interfaz de usuario mediante GUI de Matlab

Titulación: INGENIERO TÉCNICO INDUSTRIAL
Especialidad Electrónica Industria

Intensificación: AUTOMÁTICA

Alumno: VÍCTOR MANUEL LÓPEZ ROBLES

Director: JULIO JOSÉ IBARROLA LACALLE
JOSE MANUEL CANO IZQUIERDO

Cartagena, a 7 de Marzo de 2012

Contenido

CAPITULO 1: INTRODUCCIÓN Y OBJETIVOS DEL PROYECTO

1.1 Introducción

En el Departamento de Ingeniería de Sistemas y Automática de la UPCT se vienen desarrollando varios PFC mediante el kit de robótica Lego Mindstorm [1], [2].
Con este kit se pueden desarrollar maquetas que permitan visualizar problemas asociados al control de sistemas.

En estas figuras se muestran ejemplos de los tipos de maquetas que se realizan:

Figura 1.1. Montaje del NXT en forma de escorpión.

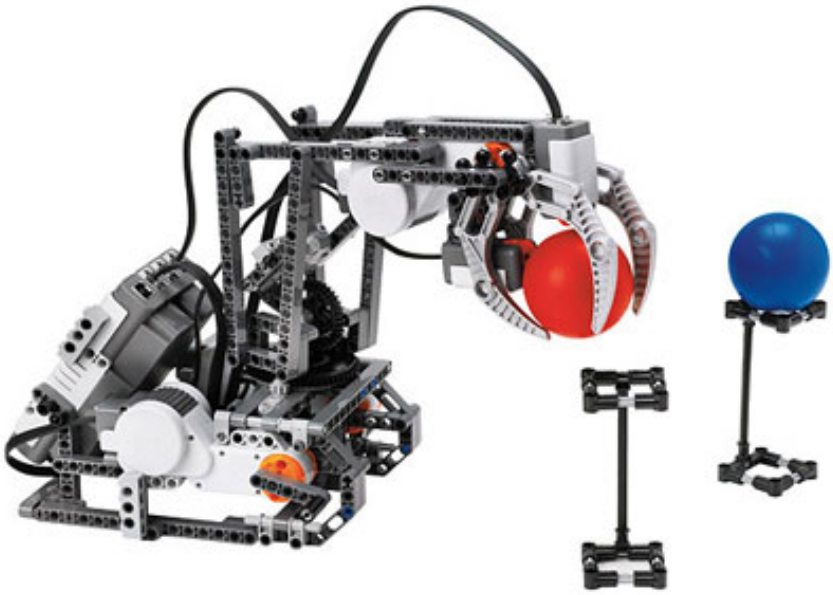


Figura 1.2. Montaje del NXT en forma de pinza.

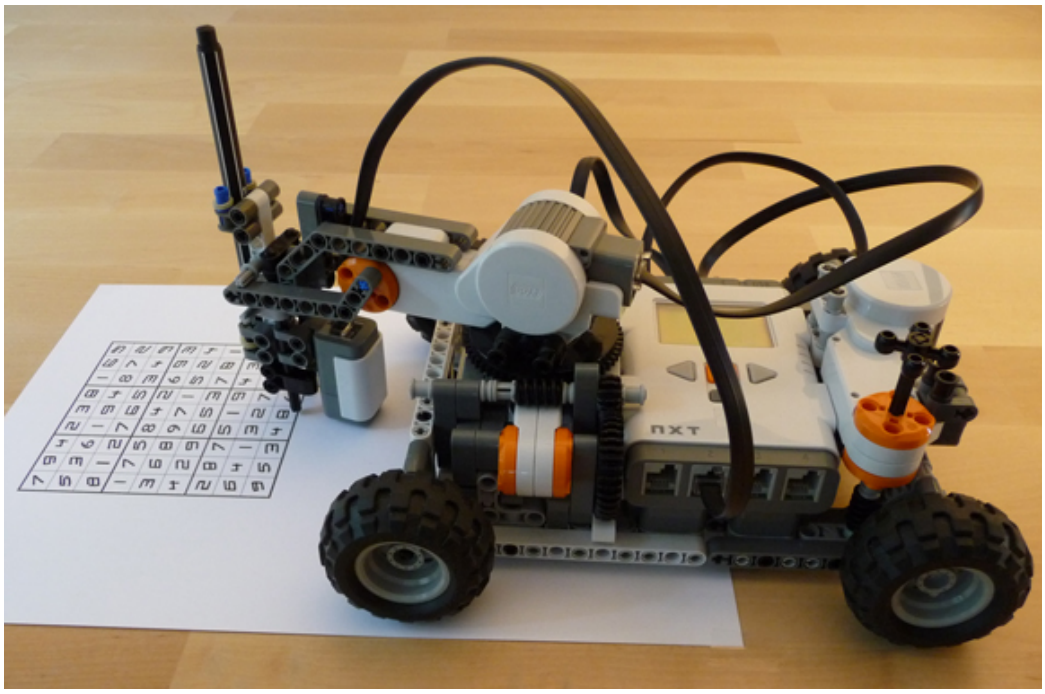


Figura 1.3. NXT resuelve sudokus.

Siguiendo la misma línea, Se pretende desarrollar una maqueta y una GUI (Graphical User Interface) asociada que permita su utilización desde un PC coordinando diferentes programas de control. El conjunto pretende ser una plataforma apta para implantar conocimientos adquiridos en asignaturas asociadas al control de sistemas.

1.2 Objetivos

Existen tres objetivos principales.

- Partiendo del prototipo construido en [1], completar la maqueta para construir un sistema independiente que sirva como banco de pruebas para realizar prácticas de control automático.

- Realizar una interfaz (GUI) en Matlab donde se muestren graficas a tiempo real de los sensores del NXT.

- Implementar un ejemplo de identificación y control del carro construido con Lego Mindstorm, siga a una pelota a lo largo de un tubo vertical con RobotC.

Estos objetivos se podrían dividir en varias tareas:

- Conexionado entre NXT y sensor de ultrasonidos.
- Adecuación de la maqueta para hacerla portable y autónoma.
- Estudio de la conexión bluetooth entre PC y NXT.
- Programación en RobotC.
- Estudio y programación de la GUI de Matlab.

- Utilización simultanea de dos programa en la realización de experimentos de la maqueta

CAPITULO 2: DESCRIPCION DEL PROTOTIPO

2.1 LEGO MINDSTORMS NXT

Para este proyecto se utilizará el kit de robótica Lego Mindstorm. Este set permite construir y programar soluciones robóticas para problemas reales. Incluye un controlador NXT programable también llamado brick, tres servomotores interactivos, sensores de ultrasonidos, sonido, luz y dos sensores táctiles, batería recargable, cables de conexión e instrucciones de construcción.

El NXT es muy utilizado en aplicaciones asociadas a la robótica ya que es fácil de manejar y programar.
Se pueden encontrar proyectos diferentes, ya sean de otras universidades como de personas que lo hacen por afición.

Figura 2.1. Ladrillo NXT.

Para la programación del NXT existen diferentes programas pero en este caso se van a utilizar dos, RobotC y Matlab.

2.1.1 DESCRIPCIÓN DEL LEGO NXT

En la figura 2.2 se muestran las distintas partes de un NXT.

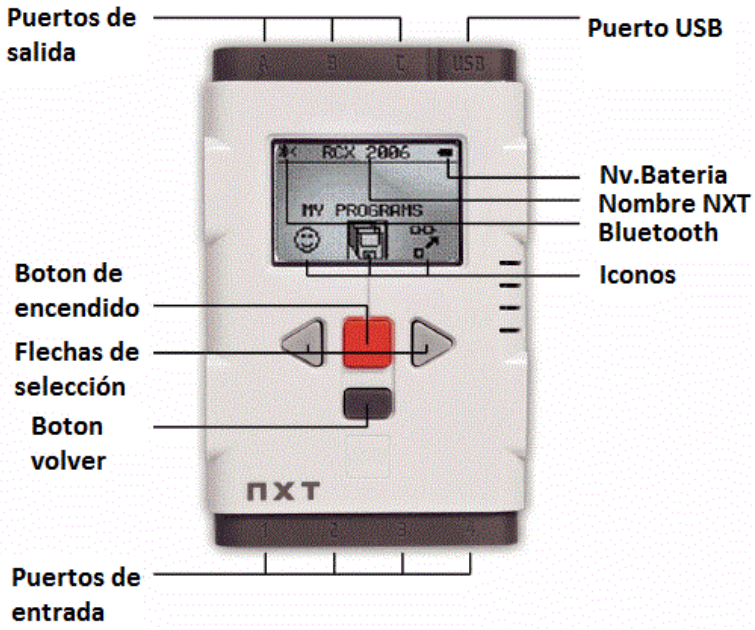


Figura 2.2 Elementos del NXT.

En la zona superior se encuentran los puertos de salida A B C. En ellos se conectan los motores que se desea controlar mediante el NXT. Estos puertos proporcionan la energía necesaria para el funcionamiento de los motores, aunque se debe tener cuidado ya que se puede producir un desgaste rápido de las pilas o de la batería.

El puerto USB sirve para conectar el NXT al PC y poder programarlo.

El nivel de batería, el nombre del NXT y los iconos, se muestran en la pantalla del NXT.

El botón naranja del centro, es el botón de encendido. Permite avanzar hacia delante en los menús.

Las flechas de selección sirven para mover el menú hacia la izquierda o derecha.

El botón volver permite retroceder en los diferentes menús del NXT.

Los puertos inferiores son los puertos de entrada. En ellos se conectan los diferentes sensores que existen para NXT.

En la parte trasera se encuentra la alimentación del NXT. Se alimenta de dos maneras, con 6 pilas AA de 1.5 voltios cada una o mediante una batería.

Figura 2.3. Batería NXT.

2.1.2 Piezas de construcción

Para construir un prototipo con un NXT, se pueden utilizar piezas de Lego. La figura 2.4 muestra las diferentes piezas que podemos encontrar en el kit de Lego.

Figura 2.4. Piezas de construcción del NXT.

Los Ejes (Axle) son elementos de diferentes tamaños (se miden en unidades de Lego) y sirven para conectar piezas que tendrán movimiento, estas pueden ser ruedas, motores, engranes o incluso vigas móviles.

Figura 2.5. Eje.

Las Vigas (Beam) son el elemento básico de construcción para los modelos de Lego. Pueden ser unidas entre sí usando clavijas o ejes.

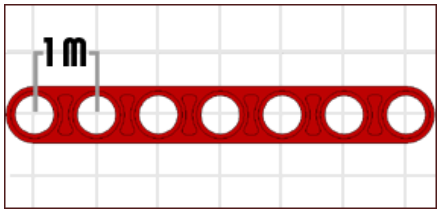


Figura 2.6. Viga.

Los Bloques (Brick) son los elementos de construcción de LEGO por excelencia, aunque se están dejando de utilizar.
El kit de Lego MINDSTORMS NXT incluye solo unos cuantos bloques.

Figura 2.7. Brick.

Las Clavijas (Connector Peg) son el elemento básico de unión. Pueden ser de diferentes tamaños y formas. La clavija básica es la mostrada en la figura 2.8.

Figura 2.8. Clavija.

Los Engranajes (gear) sirven para transferir el movimiento, incluso modificándolo en su camino. Los engranes siempre tienen un conector para un eje en medio y pueden tener conectores para clavijas alrededor. Estos se miden por el número de dientes que tienen.

Figura 2.9. Engranaje.

Las ruedas son uno de los principales componentes de Lego Mindstorms, ya que permiten que el bloque lógico pueda moverse en un espacio real e interactúe con el medio que lo rodea.

Figura 2.10. Ruedas.

2.1.3 Sensores

Existen diferentes tipos de sensores de Lego para NXT.

Sensor de ultrasonidos

Este sensor utiliza la tecnología de ultrasonidos para calcular la distancia a la que se encuentra el objeto u obstáculo.

Figura 2.11. Sensor de ultrasonidos.

Sensor de Luz

Le permite al robot distinguir entre luz y oscuridad.

Figura 2.12. Sensor de luz.

Sensor de sonido

Le permite al NXT reconocer los sonidos exteriores y así poder ejecutar las funciones establecidas en la programación.

Figura 2.13. Sensor de sonido.

Sensor de contacto

El sensor de contacto permite detectar si el bloque que lo posee ha colisionado o no con algún objeto que se encuentre en su trayectoria inmediata.

Figura 2.14. Sensor de contacto.

Los motores

Los motores de Lego generan movimiento circular.

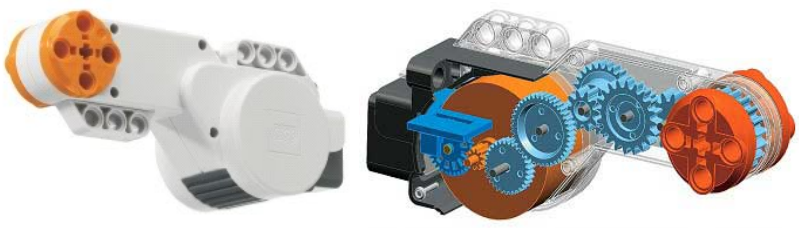


Figura 2.15. Motor.

2.1.4 Cable RJ12 para NXT

Algunas explicaciones mostradas en este apartado han sido extraídas del proyecto [1].

El cable RJ12 para NXT es un cable similar al RJ12 estándar, se diferencia por la colocación de la pestaña de enganche, que se sitúa a la derecha. Esta construcción produce problemas a la hora de buscar recambios, ya que es un conector específico. Este tipo de cable consta de 6 como puede verse en la figura 2.16.

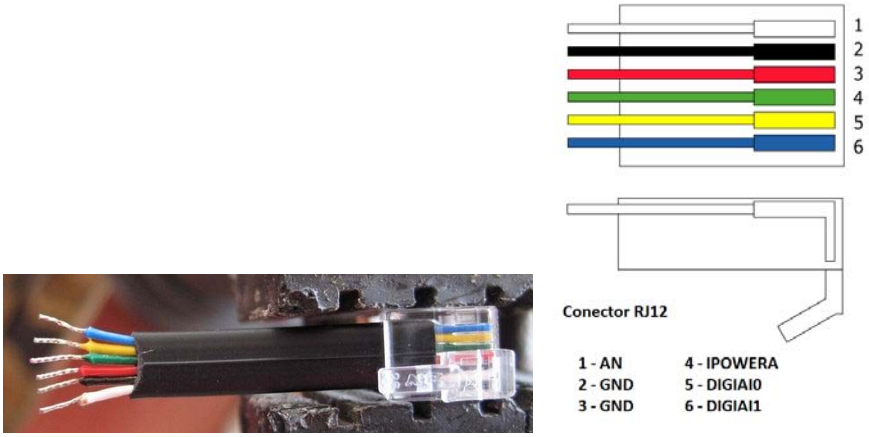


Figura 2.16. Esquema por colores.

Pin 1 (Blanco): AN

Este pin puede tener dos usos: como entrada analógica, o como fuente de energía para algunos sensores del antiguo RCX. Si se usa este pin como entrada analógica, la señal es conectada a un convertidor analógico-digital de 10 bits, incluido dentro del procesador del NXT. Para algunos sensores del RCX (también llamados sensores activos), este pin suministra una tensión teórica de 9V, correspondiente al de las pilas.

Para este tipo de sensores, el NXT ofrece una tensión durante 3ms y lee la entrada durante 0.1ms, repitiendo el ciclo indefinidamente. La frecuencia de estas lecturas es de 333Hz (Figura 2.17), y aporta una corriente aproximada de 18mA.

Figura 2.17. Tiempo de lectura.

Pines 2 y 3 (Negro y rojo): GND

Son los pines de tierra, que están conectados el uno al otro dentro del NXT y en los sensores. Las señales son medidas tomando estos pines de masa como referencia.

Pin 4 (Verde): IPOWERA

Proporciona la corriente necesaria a todos los sensores del NXT, y a los encoders de los motores. Está conectado internamente a los siete puertos de entrada y salida del brick y tiene un límite de corriente de 180mA. Eso significa que cada puerto dispone de aproximadamente de unos 25mA, aunque se puede consumir más si otro consume menos.

Pines 5 y 6 (Amarillo y azul): DIGIAI0 y DIGIAI1

Son los pines de entrada/salida usados para el protocolo de comunicación digital I2C.Se puede encontrar más información en [6]

2.2 Ventilador, bola y carro. Composición de la maqueta

Figura 2.18.Maqueta.

En la figura 2.18 se observan todos los elementos que componen la maqueta. Se observa de la maqueta dos sistemas diferentes:

Sistema Pelota:

Se trata de un sistema que pretende controlar la posición de una pelota que se encuentra dentro de un tubo en cuyos extremo hay instalados un ventilador y un sensor de ultrasonidos. Este tipo de control se realiza gracias a un NXT conectado a estos. El funcionamiento es simple, el sensor envía la medida de altura de la bola al NXT y este regula la tensión de salida que obtiene el ventilador. La medida del tubo es de 65cm de alto.

Figura 2.19 Sistema Pelota.

Sistema Carro:

El segundo sistema es el control del seguimiento de la pelota utilizando un carro formado por piezas de Lego que se mueve por el exterior de un tubo. Este sistema consta de otro NXT que está conectado por bluetooth al NXT del sistema anterior, recibiendo así la altura de la pelota y regulando la tensión de los motores del carro en función de esta. Este carro tiene unas medidas de 27 cm de ancho y 22 cm de alto, y consta de dos motores que mueven dos ruedas cada uno. El sistema incorpora un sensor de luz que resetea el encoder de los motores cuando el carro llega a 0cm. Se puede encontrar una explicación más detallada en el Anexo C3.

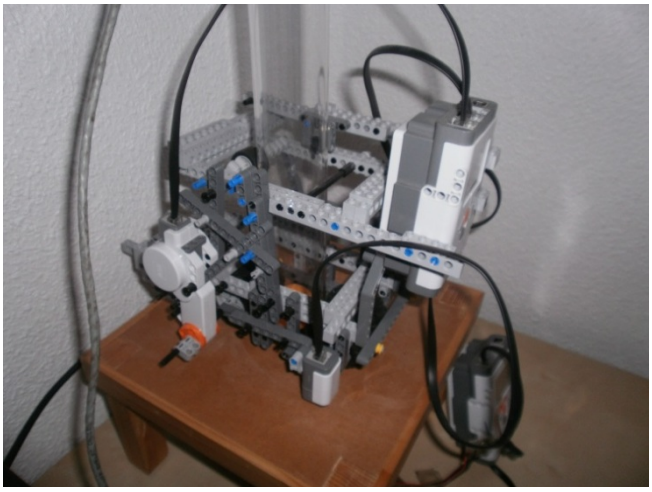


Figura 2.20 Carro vista lateral.

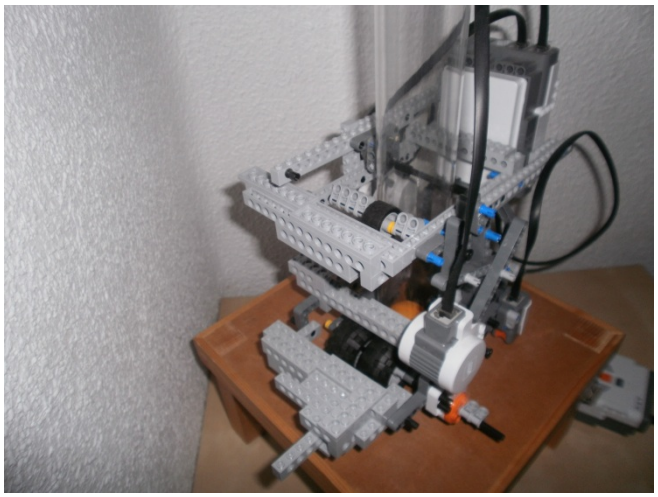


Figura 2.21 Carro vista diagonal.

CAPITULO 3: TRABAJOS PARA COMPLETAR LA MAQUETA INDEPENDIENTE.

3.1 Extender cable RJ12 para NXT

Para este proyecto se necesitan cables RJ12 que conecten los NXT con los sensores.
El problema surge cuando se quiere conectar el sensor de ultrasonidos, que se encuentra en el extremo del tubo, con el NXT que lo controla. Debido a que no se suministra un cable RJ12 tan largo, se tuvo que fabricar. Se consideraron diferentes alternativas para la fabricación de un cable RJ12 para NXT.

Soldadura.

El método de fabricación por soldadura se caracteriza por su sencillez y bajo coste.
Se pretendía unir dos cables con estaño utilizando un soldador para obtener un cable con una largura aceptable.
El inconveniente que se tiene, es el aspecto y su poca resistencia a los tirones.

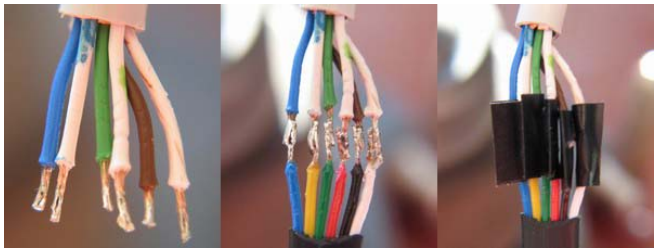
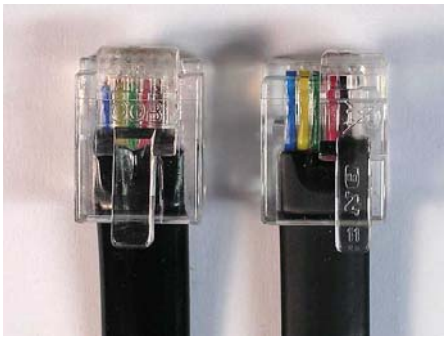


Figura 3.1. Pasos soldadura.

Fabricar cable RJ12 usando crimpadora.

Se intentó comprar los conectores RJ12 específicos y un cable de 6 pines para fabricar el cable RJ12 utilizando una crimpadora, como se puede ver en [7]. Se descartó esta opción ya que no se encontró ningún vendedor de estos conectores por ser específicos. En la figura 3.2 se puede observar la diferencia entre un cable RJ12 estándar con un RJ12 específico para NXT.



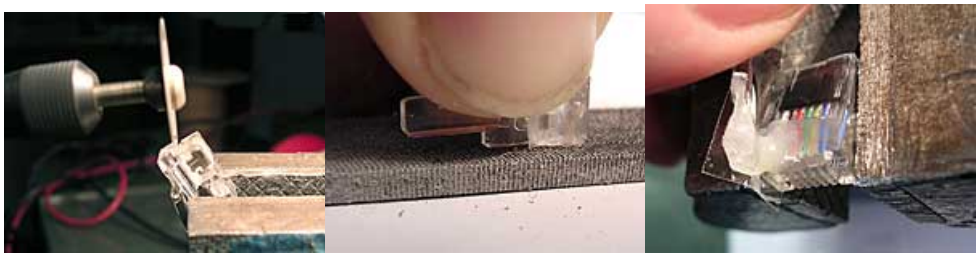
RJ12 Estándar. NXT

Figura 3.2. Comparación de cables.

Fabricar cable RJ12 a partir de un RJ12 estándar.

Con este método se pretende modificar la estructura del conector RJ12 estándar a una estructura similar de un conector RJ12 NXT. En [8] se encuentra una manera de transformar el cable RJ12 estándar en un cable RJ12 NXT. Este método se divide en tres pasos, como muestra la figura 3.3:

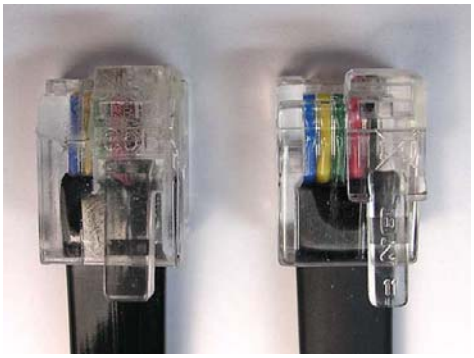
-1º Se debe cortar la pestaña del conector RJ12 con una sierra pequeña.
-2º Se lima la base para quitar las impurezas.
-3 Se pega la pestaña en la parte derecha del conector RJ12.



Cortar Limar Pegar

Figura 3.3. Pasos modificar cable estándar.

La figura 3.4 muestra la comparación de un conector RJ12 modificado mediante este método, con un conector RJ12 NXT original. Se observa la verdadera similitud.



RJ12 Modificado NXT

Figura 3.4. Comparación de cables.

Este método al contrario que los otros anteriores, es un método difícil de realizar, tiene el inconveniente de que la pestaña se puede separar.

Extender cable mediante conectores RJ12 y RJ45.

Este método encontrado en [9], se realiza utilizando un cable RJ45 y un RJ12 NXT cortado por la mitad.

Para utilizar este método es necesario:

- 1 X Cable RJ12 NXT
- 2 X Conector RJ45 Macho
- 2 X Acoplador H/H
- 1 X Crimpadora
- 1 X Cable de Red Categoría 5,5e, 6

Se Añade un conector RJ45 a las dos partes que se han cortado utilizando una crimpadora. El esquema de conexiones que debe seguir el conector es el mostrado en la figura 3.5. Se observa que el cable RJ12 consta de 6 pines y el RJ45 tiene 8 pines. Para que esto no sea un problema, se dejan sin utilizar dos pines.

Side A - Both Ends Pointing Up			-	Side B - RJ12 Facing Down		
RJ12	**Colour**	**RJ45**		**RJ12**	**Colour**	**RJ45**
1	White	1		1	White	1
2	Black	2		2	Black	2
3	Red	4		3	Red	4
4	Blue	5		4	Blue	5
5	Green	7		5	Green	7
6	Yellow	8		6	Yellow	8

Figura 3.5. Tabla de conexiones.

En la figura 3.6 se observa el resultado final.

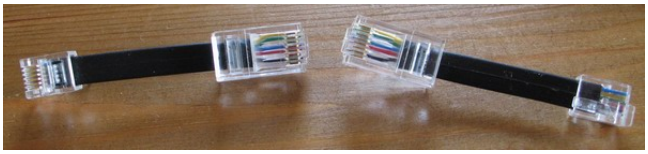


Figura 3.6. Cables RJ12/RJ45.

Una vez realizado, se debe comprar o fabricar un cable RJ45 sin cruzar.

Por último, se utilizan los dos acopladores para unir los tres cables, dando como resultado un cable RJ12 extenso y aceptable para nuestra maqueta como se observa en la figura 3.7.

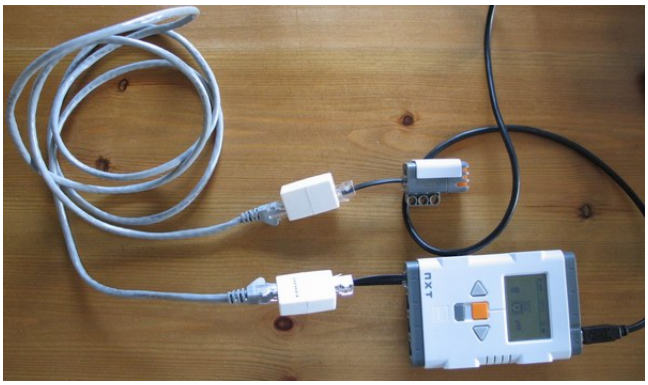


Figura 3.7. Resultado obtenido.

Como se puede ver, este método tiene una apariencia más agradable a la vez que es más resistente a tirones por lo que finalmente optamos por esta solución.

3.2 Fuente de alimentación portable.

La maqueta al no disponer de una fuente de alimentación propia, dependía de una fuente de un laboratorio. Para solucionar este problema y tener una maqueta portable, se tuvo que añadir una fuente de alimentación portable encontrada en el laboratorio.

Las características de la fuente de alimentación de laboratorio son:
V=7.6 voltios.
I=180 miliamperios.

Las características de la nueva fuente de alimentación portable son:
V=9 voltios.
I=200 miliamperios.

Esta diferencia de voltaje provocaba un cambio en el sistema. El controlador PID que llevaba incluido no funcionaba de la misma manera y se tuvieron que calcular otros parámetros para el PID. Esta fuente finalmente, fue pegada con velcro en la zona inferior de la maqueta.

Figura 3.8. Fuente en maqueta.

3.3 Mejorar estabilidad física del carro.

En un principio el carro disponía de tres ruedas. Los motores transmitían la fuerza producida a estas ruedas utilizando engranajes.
Estos engranajes no estaban bien colocados, ya que se salían mientras el carro funcionaba.

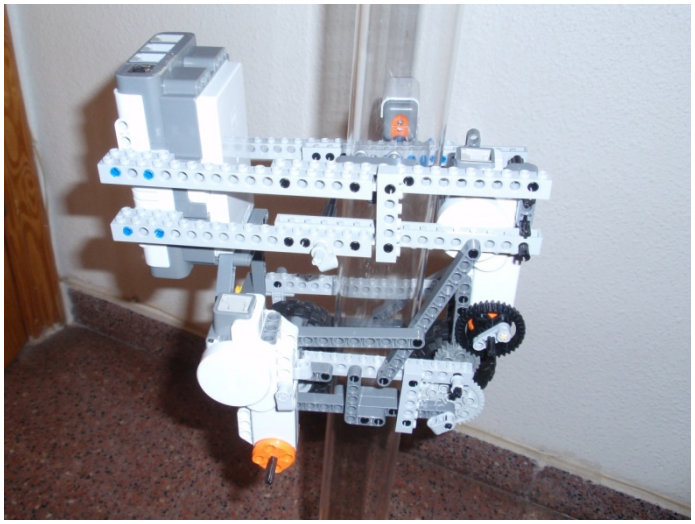


Figura 3.9. Carro vista lateral.

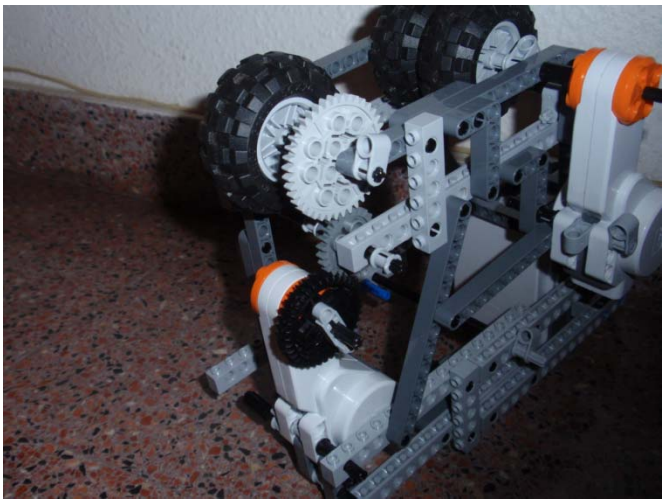


Figura 3.10. Carro vista inferior.

Al añadirle una cuarta rueda y reforzar la base del carro se mejoró este aspecto por completo, consiguiendo una estructura más fiable. Para ello se cambió el motor de sitio, junto al eje de las ruedas, ya que uno de los principales problemas eran los engranajes y por ello se han suprimido. En su lugar, se colocó una rueda pequeña para que no exista demasiada fuerza de rozamiento entre el tubo y el carro.

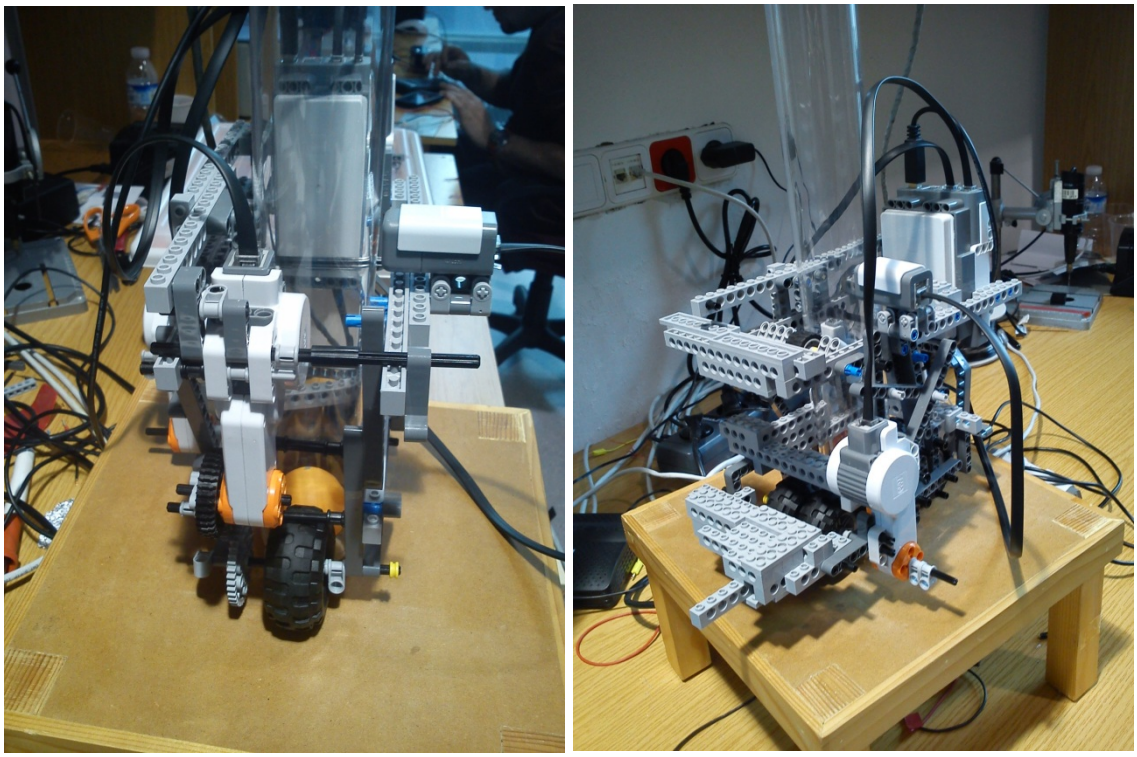


Figura 3.11. Comparación del carro antes y después de su modificación.

Finalmente para solucionar el problema que tiene el carro al bajar por el tubo explicado en el Anexo C.3 se cambió el sensor de luz de posición como se puede ver en la figura 3.14.

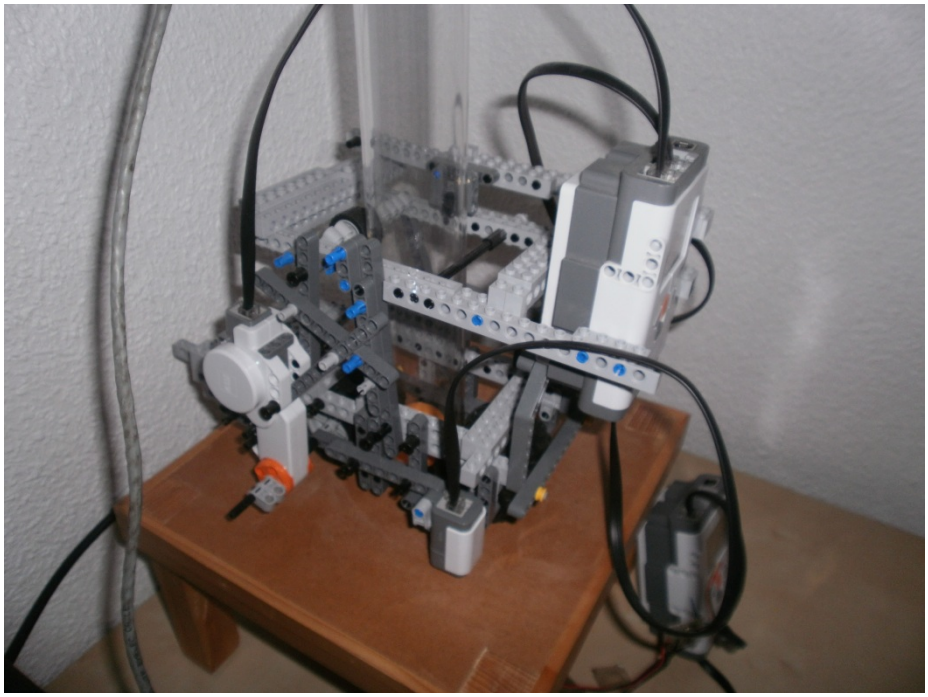


Figura 3.12. Carro final.

3.4. Soporte para NXT

La maqueta no disponía de un soporte para colocar el NXT que controla la pelota por lo que se tuvo que fabricar uno. Se utilizaron escuadras metálicas para su construcción. Una vez terminado, se añadió a la maqueta como muestra la figura 3.15.

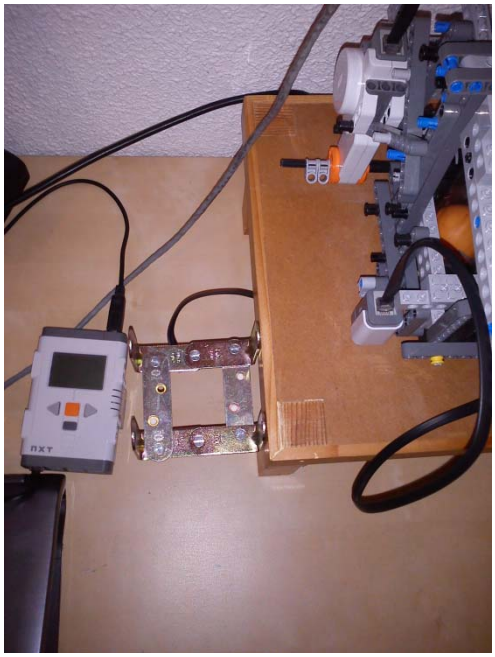


Figura 3.13 Soporte

CAPITULO 4: SOFTWARE NECESARIO Y FUNCIONAMIENTO DE LA GUI DE MATLAB

4.1 ROBOTC

RobotC es un programa en el que se programa en un tipo de C especifico, como su nombre indica, el cual instalado en el PC, permite escribir el código de programación que vamos a utilizar para el manejo del NXT y descargarlo en su memoria interna. Para ello es necesario instalar un firmware específico creado por RobotC en el NXT

4.1.1 Entorno de programación

Se van a explicar las partes del entorno de programación, destacando las más importantes.

En la figura 4.1, se observa el entorno de programación.

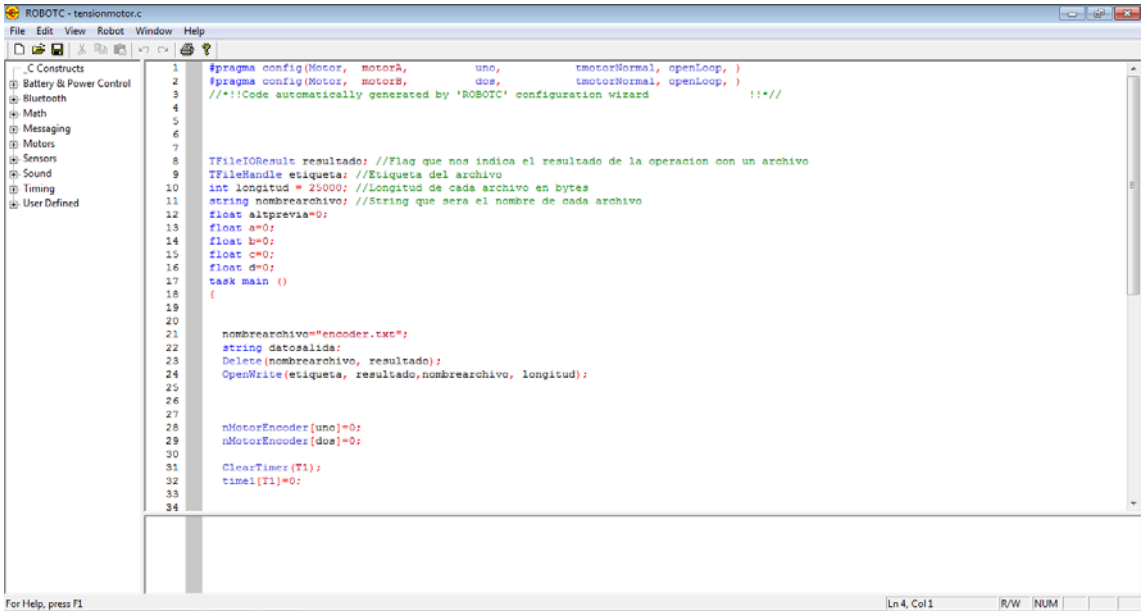


Figura 4.1. Entorno de programación RobotC.

En la parte izquierda destaca la ventana adicional con las plantillas de todas las funciones disponibles para RobotC, ordenadas por categorías y contienen la explicación inmediata de su funcionamiento al hacer doble clic en alguna de ellas. A su vez, la instalación suministra una carpeta con gran cantidad de programas de ejemplo, ideales para familiarizarse con el lenguaje. En el menú superior encontramos los distintos menús del programa. Las opciones más útiles aparecen en la pestaña Robot. En la figura 4.2 se ve el contenido de esta pestaña:

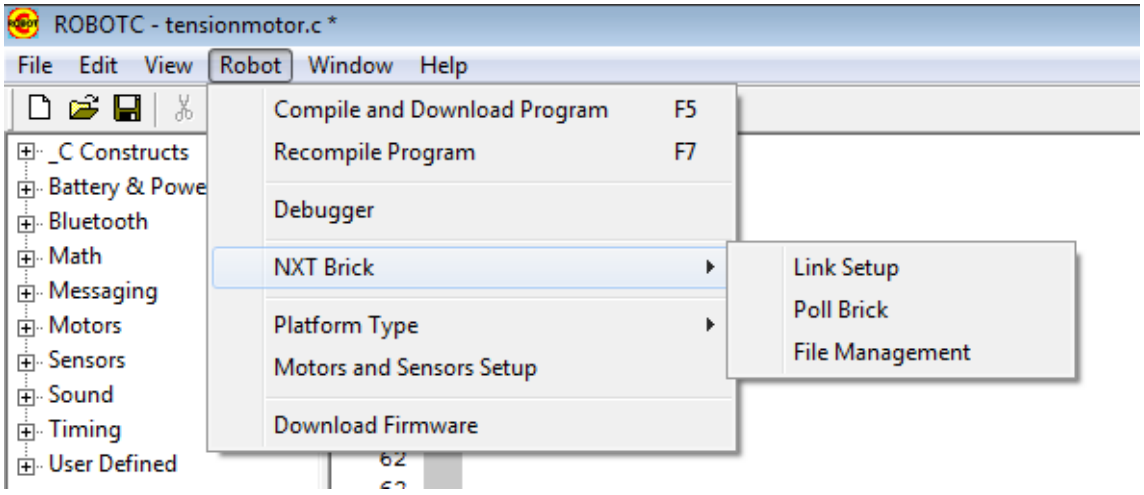


Figura 4.2. Menú "Robot" de RobotC.

Las opciones que aparecen son:

- Descargar programa al ladrillo.
- Compilar, o en otro caso, recompilar el código.
- Debugger.
- NTX Brick que incluye la conexión con el ladrillo y el administrador de fichero.
- Tipo de plataforma para elegir entre las distintas versiones del ladrillo.
- También podemos hacer un setup de los motores y sensores para asignar a cada puerto del ladrillo el elemento que irá conectado.
- Descargar el Firmware al ladrillo para poder utilizar RobotC en él.

En cuanto al espacio de trabajo, se nos facilita la escritura por la numeración de líneas, el uso de distintos colores en función del tipo de palabra, y el uso de marcadores. También resulta útil la opción de autocompletado mientras se escribe el programa.

4.1.2 Modos de conexión

Para poder descargar el firmware y descargar el programa, es necesario tener una conexión entre el ordenador y el ladrillo. Por ello es necesario hacer una explicación rápida de los dos modos de conexión disponibles, USB o Bluetooh.
Para configurar el modo de conexión, accedemos a la pestaña robot, luego NXT Brick y link setup. Una vez hecho esto, nos aparece la siguiente ventana (Figura 4.3):

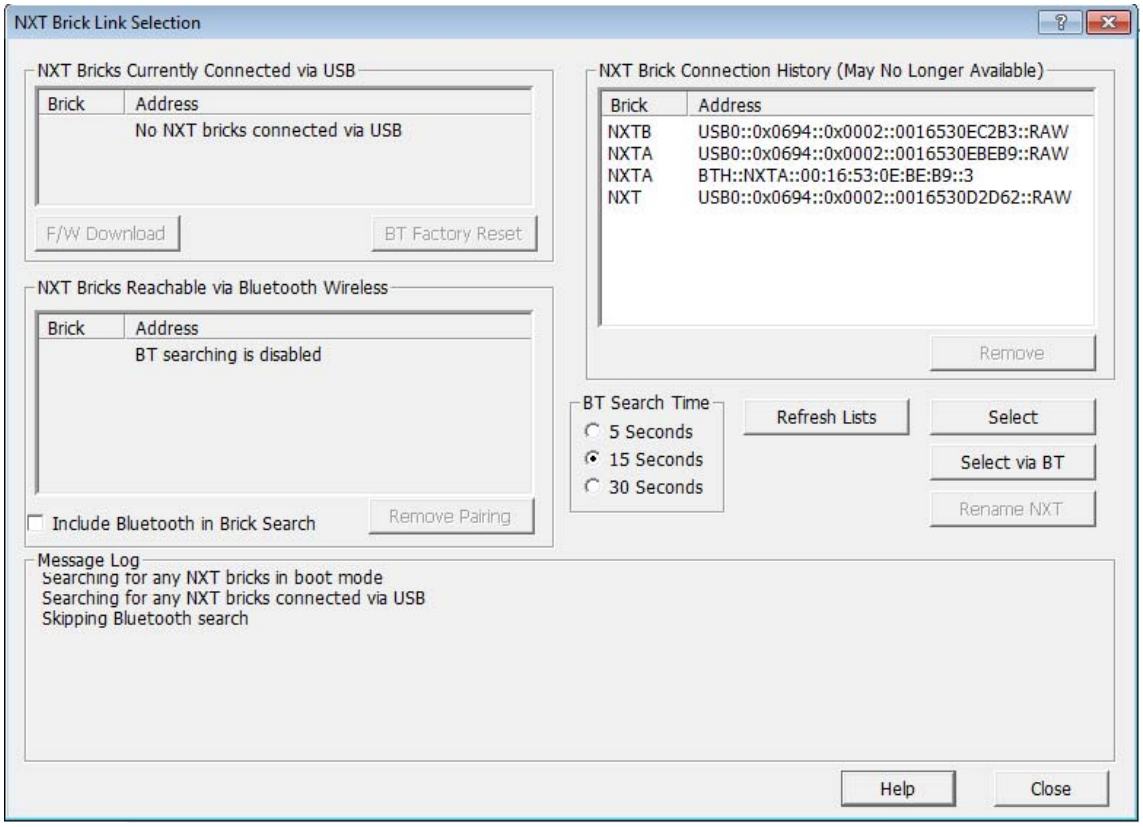


Figura 4.3. Ventana del asistente de conexión de RobotC.

Si tuviéramos conectado el brick por USB aparecería en la parte superior izquierda y si fuera mediante por Bluetooh aparecería en parte central izquierda. Una vez que se sepa la conexión que se va a utilizar, pinchamos sobre él y le damos a Select, situado en la parte derecha central. Un pitido nos informara que la conexión es correcta.
Es necesario descargar el Firmware, preferiblemente se debe de realizar mediante cable USB, y más tarde, ya sería posible utilizar la conexión Bluetooth. Mediante esta conexión Bluetooh se puede realizar las siguientes funciones:

- Descargar el código C desde el RobotC hasta el ladrillo del NXT.
- Descargar los ficheros *.txt que contienen los resultados de los experimentos.
- Interactuar en tiempo real y de forma inalámbrica sobre el ladrillo sin necesidad de tocar el ladrillo físico.

4.1.3 Entorno de ejecución. Debugger

Una vez que tenemos el programa descargado en nuestro ladrillo, aparece en la pestaña de Robot, un nuevo menú desplegable llamado Debug Windows. Este menú nos permite mostrar en pantalla diferentes cuadros para operar u observar cómo funciona nuestro ladrillo. Si activamos todas las opciones el aspecto del entorno de trabajo se puede ver en la figura 4.4.

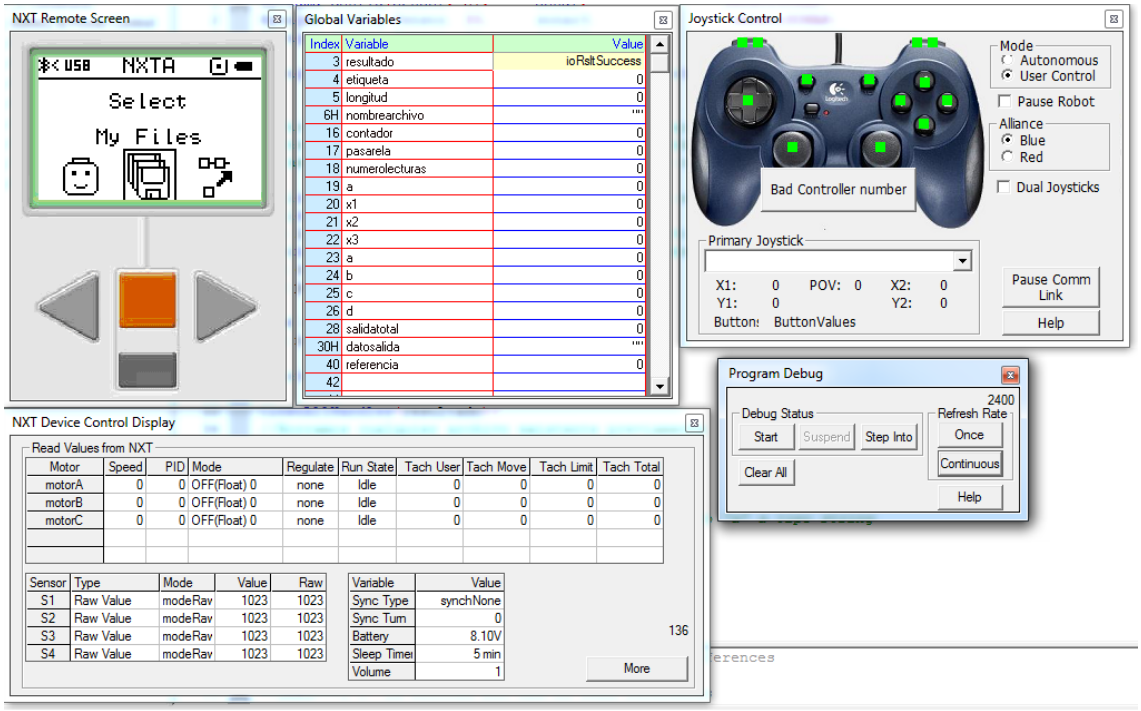
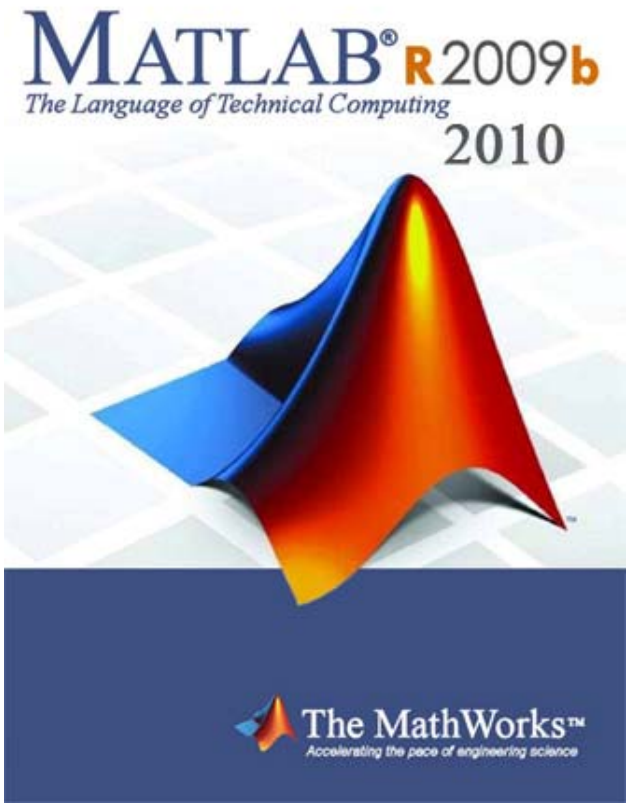


Figura 4.4. Opciones seleccionables del debugger de RobotC.

Si comenzamos a enumerar y explicar de arriba abajo y de izquierda a derecha tenemos:

- NXT Remote Screen: permite operar sobre el ladrillo sin tener que irnos al ladrillo físico.
- Global Variables: muestra el valor en tiempo real de todas las variables escritas en nuestro código.
- System Parameters: nos dice algunos parámetros internos del ladrillo.
- Program Debug: Es la ventana básica para poner en marcha la ejecución del programa.
- NXT Device Control Display: permite ver y modificar el valor de los elementos conectados al ladrillo, ya sean motores o sensores.
- Joystick Control: actúa a modo de mando a distancia.
- Datalog: guarda un registro de valores.

4.2. MATLAB



MATLAB es un software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows y Apple Mac OS X. Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware.

El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (toolboxes); y las de Simulink con los paquetes de bloques (blocksets). Es un software muy usado en universidades y centros de investigación y desarrollo.

Para el desarrollo de la maqueta se ha utilizado Matlab con RWTH - Mindstorms NXT Toolbox for MATLAB. Esta toolbox facilita la comunicación con el NXT, ya que ha sido creada concretamente para ello.

4.2.1 Entorno de programación

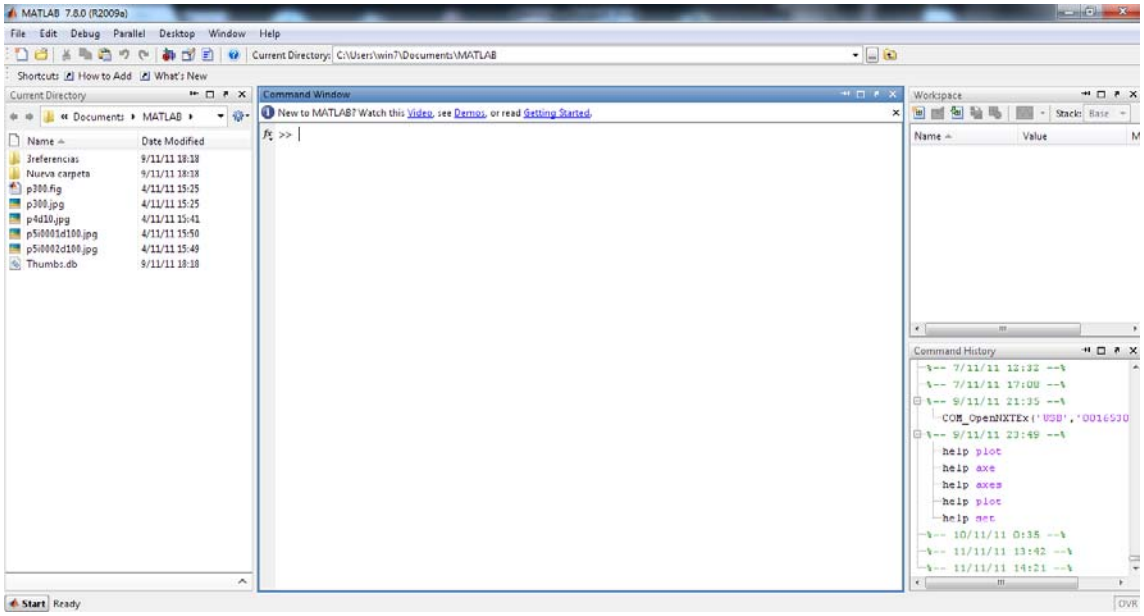


Figura 4.5. Interfaz Matlab.

Al iniciar Matlab, aparece su interfaz mostrada en la Figura 4.5.

La parte más importante es la ventana Command Window, situada en el centro.
En esta sub-ventana se escriben y ejecutan los distintos comandos que utiliza Matlab.
Cuanto muestra (>>), indica que el programa está preparado para recibir instrucciones.

En la parte izquierda aparece la ventana Current Directory, que muestra los ficheros del directorio activo o actual. El directorio activo se puede cambiar desde Command Window, o desde la propia ventana con los métodos de navegación de directorios propios de Windows. Clicando dos veces sobre alguno de los ficheros *.m del directorio activo se abre el editor de ficheros de MATLAB, herramienta fundamental para la programación. A la derecha aparece el Workspace, que contiene información sobre todas las variables que se hayan definido en esta sesión y permite ver y modificar las matrices con las que se esté trabajando.
En la parte inferior derecha aparece la ventana Command History, que muestra los últimos comandos ejecutados en Command Window. Estos comandos se pueden volver a ejecutar haciendo doble clic sobre ellos.

En la parte inferior izquierda de la pantalla aparece el botón Start. Da acceso inmediato a ciertas capacidades del programa.
La Figura 4.6 muestra las posibilidades de Start/MATLAB, mientras que la Figura 4.7, las opciones de Start/Desktop Tools. Estas permiten el acceso a los principales componentes o módulos de MATLAB.

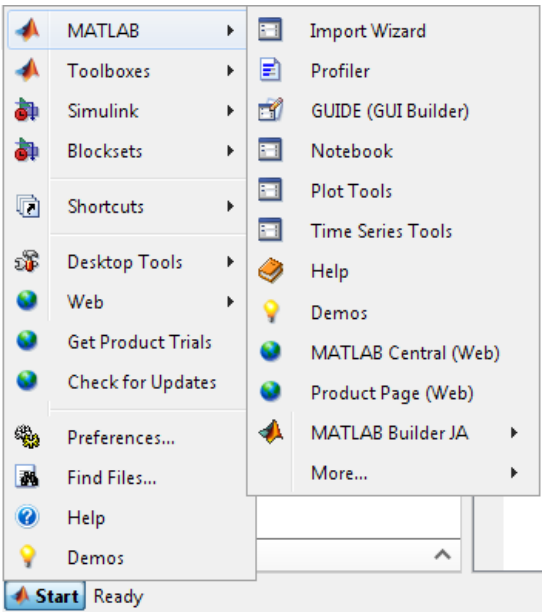


Figura 4.6. Menú Matlab.

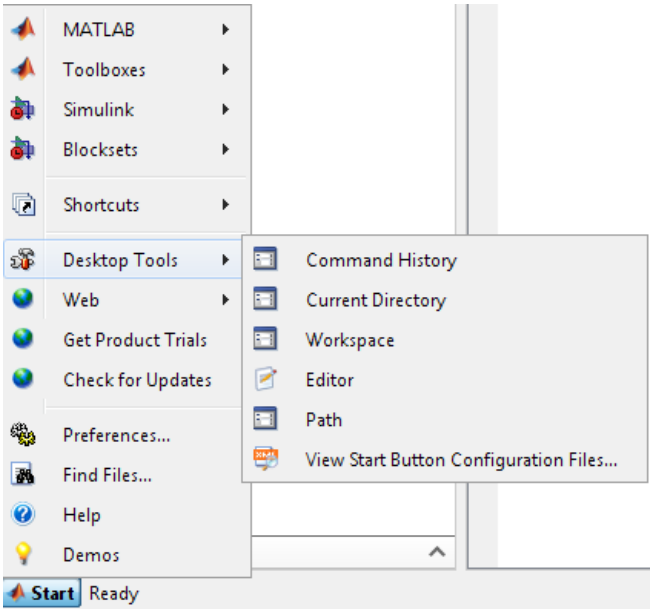


Figura 4.7. Menú Desktop Tools.

4.3 GUIDE

Las interfaces gráficas de usuario (GUI - Graphical User Interface), es la forma en que el usuario interactúa con el programa o el sistema operativo de una computadora. La forma de implementar las GUI con Matlab es crear los objetos y definir las acciones que cada uno va a realizar. Al usar GUIDE obtendremos dos archivos:

- Un archivo FIG: Contiene la descripción de los componentes que contiene la interfaz.
- Un archivo M: Contiene las funciones y los controles del GUI así como el callback.

Un callback se define como la acción que llevará a cabo un objeto de la GUI cuando el usuario lo active. Para ejemplificarlo, si se crea una ventana con un botón el cual al presionarlo ejecuta una serie de acciones, estas representan la función callback del botón.

4.3.1 Interfaz.

Para crear una GUI en Matlab usamos GUIDE. Se teclea guide en la ventana de comandos o en el menú principal en File > New > GUI como muestra la figura 4.8.

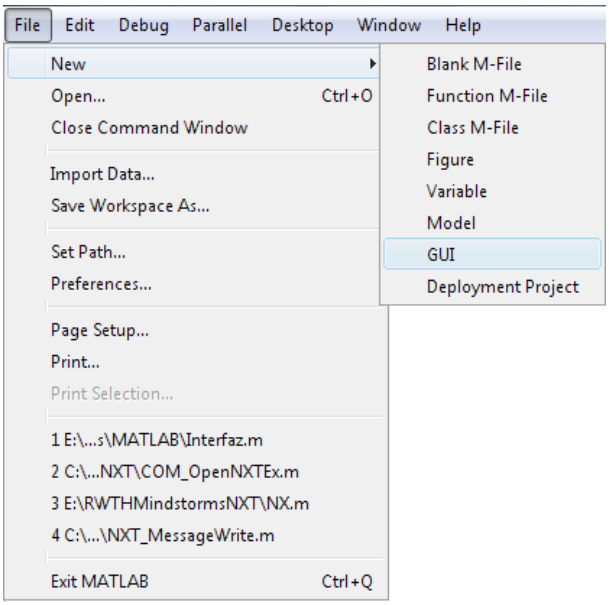


Figura 4.8 Menú File.

Se presenta el siguiente cuadro de diálogo mostrado en la figura 4.9.

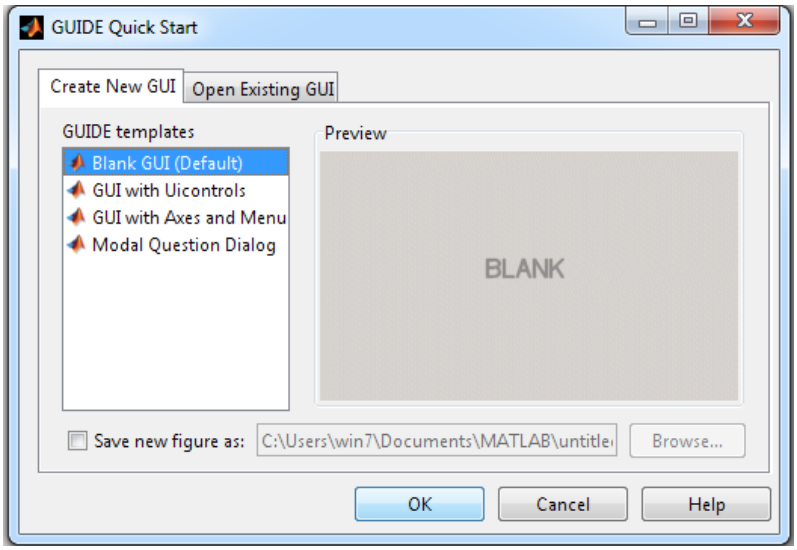


Figura 4.9. Ventana de inicio de GUIDE.

Se presentan las siguientes opciones:

a) Blank GUI (Default)
La opción de interfaz gráfica de usuario en blanco (viene predeterminada), nos presenta un formulario nuevo, en el cual podemos diseñar nuestro programa.

b) GUI with Uicontrols
Esta opción presenta un ejemplo en el cual se calcula la masa, dada la densidad y el volumen, en alguno de los dos sistemas de unidades. Podemos ejecutar este ejemplo y obtener resultados.

c) GUI with Axes and Menu
Esta opción es otro ejemplo el cual contiene el menú File con las opciones Open, Print y Close. El formulario contiene un *Popup menu*, un *push button* y un objeto *Axes*. Se puede ejecutar el programa eligiendo alguna de las seis opciones que se encuentran en el menú despegable y haciendo clic en el botón de comando.

d) Modal Question Dialog
Con esta opción se muestra en la pantalla un cuadro de diálogo común, el cual consta de una pequeña imagen, una etiqueta y dos botones *Yes* y *No*. Dependiendo del botón que se presione, el GUI retorna el texto seleccionado (la cadena de caracteres 'Yes' o 'No').

Eligiendo la primera opción, *Blank GUI*, aparece la ventana mostrada en la figura 4.10.

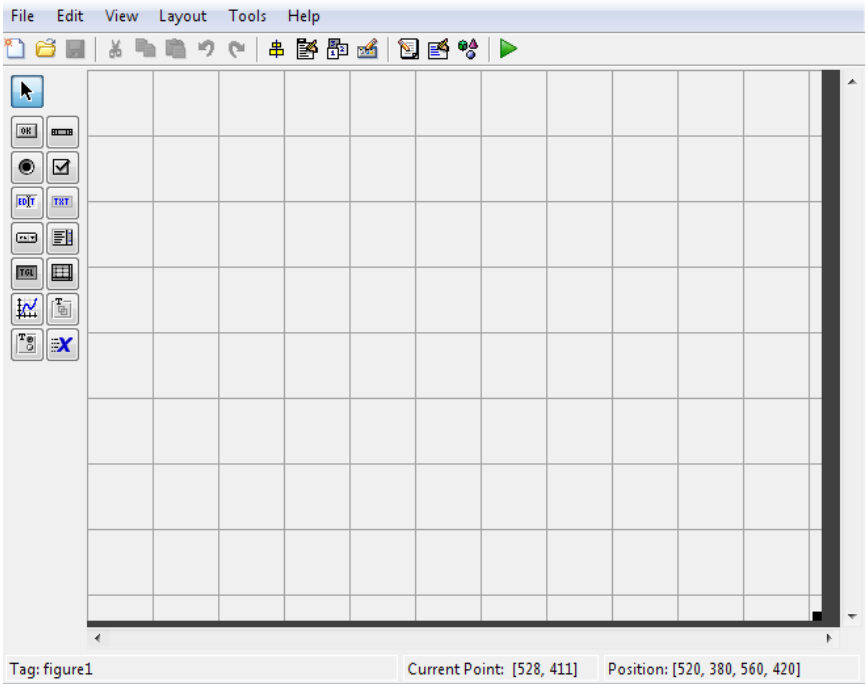


Figura 4.10. Interfaz GUIDE.

En la parte superior se encuentran los menús y opciones de GUIDE. En la zona izquierda se aprecian los diferentes controles que se utilizan. En la parte central, el área de diseño donde se colocan los elementos a usar.

Los elementos más importantes de GUIDE son:

- **De la barra superior:**

Inspector de propiedades. Cada control cuenta con diferentes propiedades y es aquí donde se pueden cambiar: el color, el nombre, el tag, el valor, el callback entre otros.

Activar Figura. Permite probar la GUI creada.

- **De la barra izquierda:**

Push Button. Crea un botón.

Un botón “Push Button” genera una acción cuando se presiona.

Radio Button. Crea un botón circular.

Edit Text. Crea un campo de texto.

Menu Pop-up: Se abren para desplegar una lista de opciones cuando el usuario hace clic sobre la flecha que se encuentra adjunta. Las opciones se introducen en la propiedad "String", una por línea en la caja de edición con el mismo título.

Axes. Crea un área para gráficas.
El botón "Axes" permite a una GUI desplegar gráficos e imágenes. Como todo objeto gráfico, el componente "Axes" tiene propiedades que pueden controlar muchos aspectos de su comportamiento y apariencia.

Static Text. Crea un letrero.

Los controles "Static Text" despliegan líneas de texto. El botón "Static Text", usualmente, se utiliza para colocar la etiqueta de otros controles, suministrar direcciones al usuario, o indicar valores asociados con un objeto deslizable. El usuario no puede cambiar su texto interactivamente y no existe una forma de invocar la rutina callback asociada con él.

4.3.2 Propiedades de los componentes.

Cada uno de los elementos de la GUI, tiene un conjunto de opciones accesibles con el clic derecho. En la figura 4.11 Se muestran todas estas opciones.

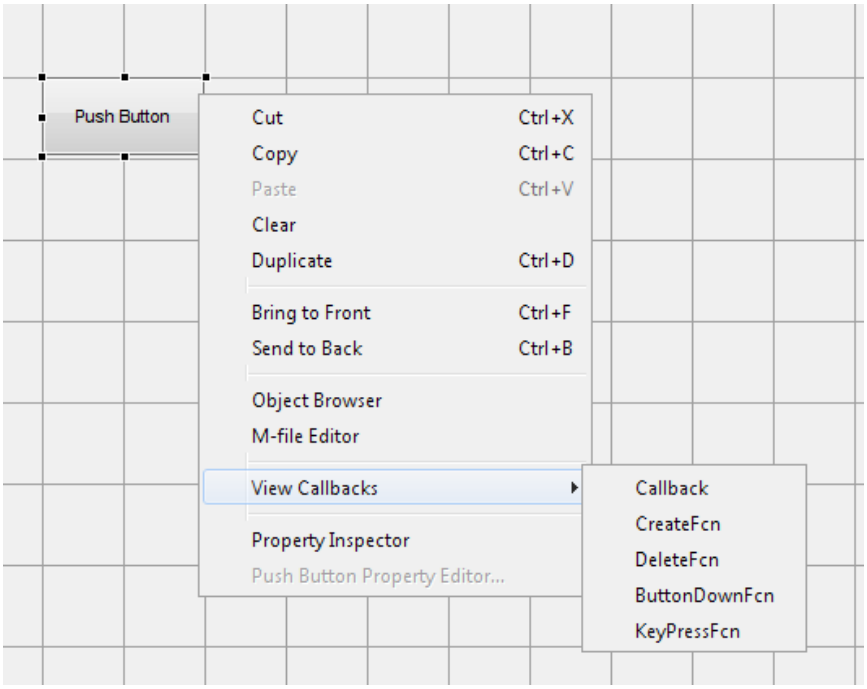


Figura 4.11. Opciones del componente.

Si se elige la opción Property Inspector, aparece la ventana mostrada en la figura 4.12. Permite personalizar las características de cada elemento.

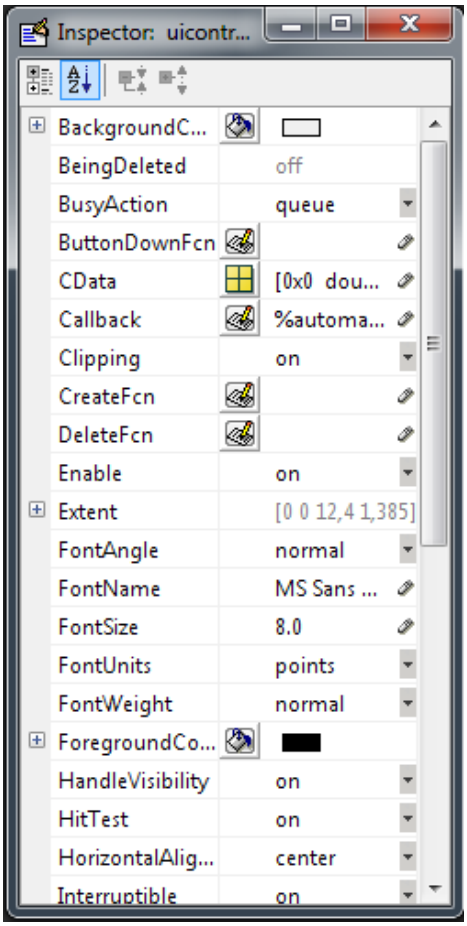


Figura 4.12. Entorno Property Inspector.

Las propiedades varían dependiendo del objeto que se va a utilizar.
Las más comunes son:

•Background Color: Cambia el color del fondo del objeto.

•CallBack. Le dice al objeto que hacer cuando este se active.

•Enable: Habilita o deshabilita el objeto.

•String: Es el texto que muestra el objeto en el caso de botones, cajas de texto, texto estático.

•Tag: Identifica al objeto.

Al hacer clic derecho en el elemento ubicado en el área de diseño, una de las opciones más importantes es *View Callbacks*. Al ejecutarla, abre el archivo *.m* asociado al diseño y se posiciona en la parte del programa que corresponde a la subrutina que se ejecutará cuando se realice una determinada acción sobre el elemento que estamos editando. Por ejemplo, al ejecutar *View Callbacks>>Callbacks* en el *Push Button*, se ubica en la parte del programa:

```
function pushbutton1_Callback(hObject, eventdata, handles)
```

Si se selecciona M-file editor, se muestran todas las funciones de la GUI. Destacando las dos más importantes:

•La función "*Opening function*" desarrolla tareas antes de que la GUI sea visible al usuario, tales como la creación de datos para la GUI. Por defecto, se denomina a esta función como *untitled1_OpeningFcn*, siendo untitled1 el nombre de la GUI que el usuario puede cambiar.

• La función "*Output function*" da salida a las variables hasta la línea de comandos.

4.3.3 Funcionamiento de la GUIDE

•Manejo de datos entre los elementos de la aplicación y el archivo .m

Todos los valores de las propiedades de los elementos (color, valor, posición, string…) y los valores de las variables transitorias del programa se almacenan en una estructura, los cuales son accesibles mediante un único y mismo *identificador* para todos éstos.

```
handles.output = hObject;
```

handles, es el identificador de los datos de la aplicación. Esta definición de identificador es salvada con la siguiente instrucción:

```
guidata(hObject, handles);
```

guidata, es la sentencia para guardar los datos de la aplicación.

Aviso: *guidata* es la función que guarda las variables y propiedades de los elementos en la estructura de datos de la aplicación, por lo tanto, como regla general, en cada subrutina se debe escribir en la última línea lo siguiente:

```
guidata(hObject,handles);
```

Esta sentencia garantiza que cualquier cambio o asignación de propiedades o variables quede almacenado.

Por ejemplo, si dentro de una subrutina una operación dio como resultado una variable *pfc* para poder utilizarla desde el programa u otra subrutina se debe guardar de la siguiente manera:

```
handles.pfc=pfc;
guidata(hObject,handles);
```

La primera línea crea la variable *pfc* a la estructura de datos de la aplicación apuntada por *handles* y la segunda graba el valor.

•Sentencias GET y SET

La asignación u obtención de valores de los componentes se realiza mediante las sentencias *get* y *set*. Por ejemplo si se quiere que la variable *utpl* tenga el valor del *Slider* se escribe:

```
utpl= get(handles.slider1,'Value');
```

Siempre se obtienen los datos a través de los identificadores *handles*.

Para asignar el valor de la variable *utpl* al *statictext* identificado como *text1* se utiliza el siguiente código:

```
set(handles.text1,'String',utpl);
```

•Mensajes de usuario

Existen diferentes tipos de mensajes destinados al usuario. Se muestra su código y en las figuras su ejecución.

```
warndlg('Esto es un aviso','Curso_GUIDE');
```

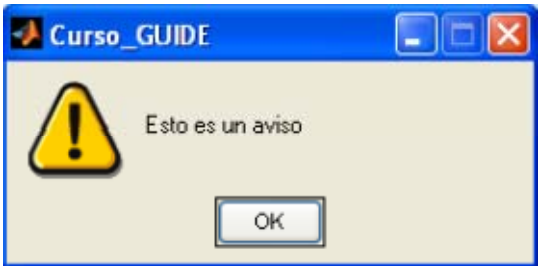


Figura 4.13. Ventana de aviso.

```
errordlg('Esto es un mensaje de error',' Curso_GUIDE ');
```

Figura 4.14. Ventana de error.

helpdlg('Esto es una ayuda',' Curso_GUIDE ');

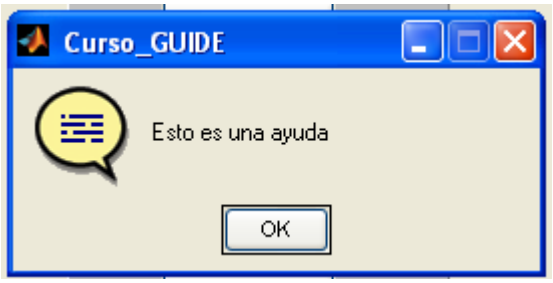


Figura 4.15. Ventana de ayuda.

msgbox('Esto es un cuadro de mensaje',' Curso_GUIDE ');

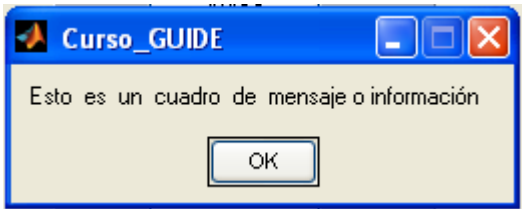


Figura 4.16. Ventana de mensaje.

questdlg('Esto es una pregunta',' Curso_GUIDE ');

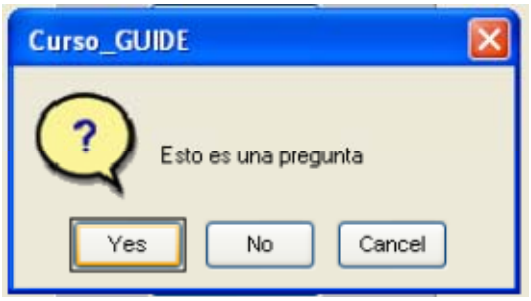


Figura 4.17. Ventana de pregunta.

inputdlg('DATO','Encabezado');

Figura 4.18. Ventana de introducción de datos.

Para el caso especial de las preguntas, se puede ejecutar o no sentencias dependiendo de la respuesta escogida. Por ejemplo, si se desea salir o no del programa. Se escribe:

```
opc=questdlg('¿Desea salir del programa?','SALIR','Si','No','No');
if strcmp(opc,'No')
return;
```

```
end
clear,clc,close all
```

Figura 4.19. Ventana de pregunta.

La función strcmp compara dos strings y si son iguales, retorna el valor 1 (true).
Clear elimina todos los valores de workspace, clc limpia la pantalla y close all cierra todas las GUI. Nótese que la secuencia 'Si','No','No' termina en 'No'. Con esto se logra que la parte No del cuadro de pregunta esté resaltado. Si terminara en 'Si', la parte Si del cuadro de pregunta se resaltaría.

•Axes (Gráficas)

Para trabajar con un Axes se necesita seleccionarlo previamente. Para entender su funcionamiento se utiliza el siguiente ejemplo:

```
x=[1 2 3 4 5 6]
y=[1 2 3 4 5 6]
axes(handles.axes1);
handles.h=plot(x,y);
title('Pelota') % Pone un título
xlabel('Tiempo') % Etiqueta el eje horizontal
ylabel('Altura') % Etiqueta el eje vertical
legend('Altura bola', 'referencia') % Pone una leyenda
grid on;
set(handles.h,'Color','r');
```

Si se desea establecer una imagen o grafica en el axe1 se debe seleccionar previamente ese axes:

```
axes(handles.axes1);
```

Una vez seleccionado, se identifica la gráfica con:

```
handles.h=plot(x,y);
```

A continuación, se añade el título, leyenda, etc.

```
title('Pelota') % Pone un título
xlabel('Tiempo') % Etiqueta el eje horizontal
ylabel('Altura') % Etiqueta el eje vertical
legend('Altura bola', 'referencia') % Pone una leyenda
```

grid on;
Por último, se asigna mediante su identificador, la gráfica al axes1 con el tipo de características que se desean. En este caso, línea de color rojo.

set(handles.h,'Color','r');

El resultado se observa en la figura 4.20.

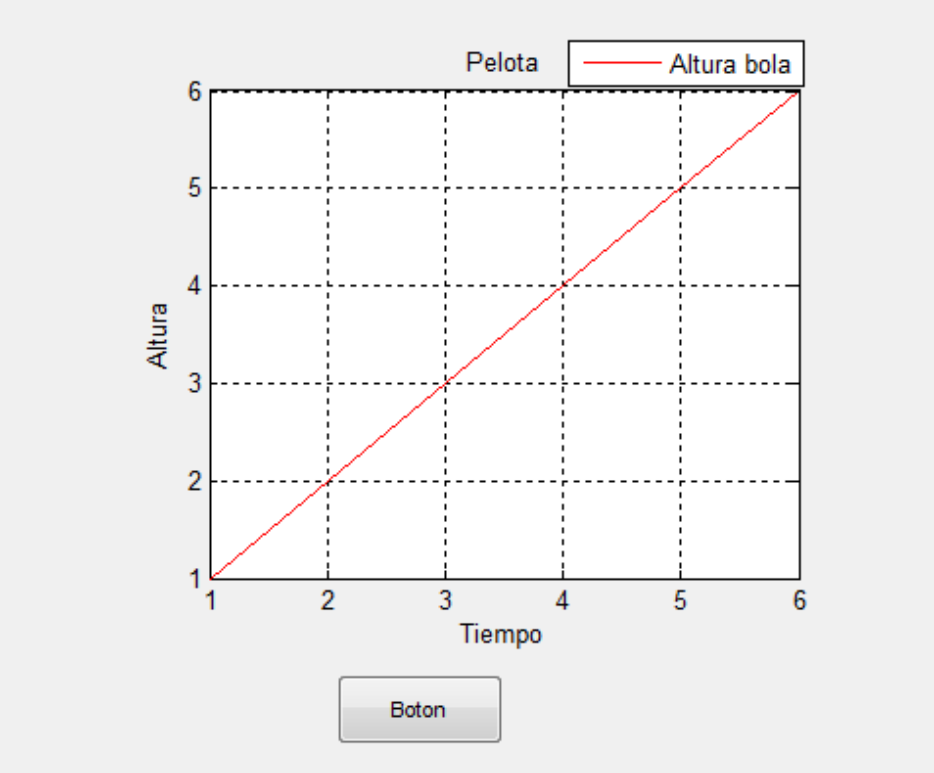


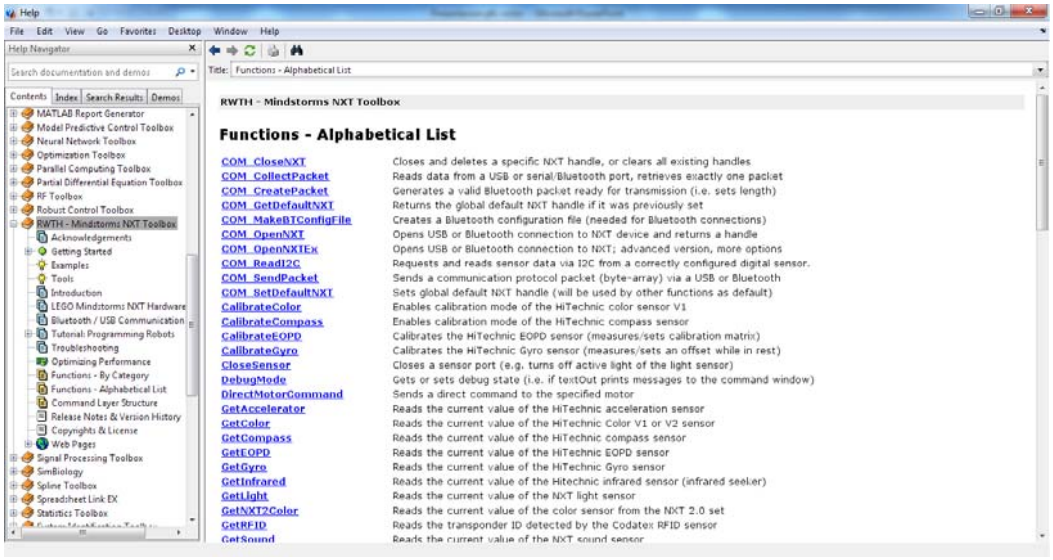
Figura 4.20. Ejemplo de Axes1.

•PopupMenu

Para trabajar con un menú popup, se utiliza el siguiente código.

```
function popupmenu1_Callback(hObject, eventdata, handles)
  switch get(handles.popupmenu1,'Value') %Obtiene el valor elegido
      case 2 %Código del caso 2
          set(handles.edit4,'Enable','on');
      case 3 % Código del caso 3
          set(handles.edit4,'String','0');
        set(handles.edit4,'Enable','off');
```

4.4 Instalación del toolbox necesario



4.4.1 Introducción.

Esta información fue sacada del proyecto [2].

El toolbox RWTH ha sido desarrollado para el control de LEGO Mindstorms NXT mediante Matlab. La conexión PC-NXT se realiza a través de una conexión inalámbrica Bluetooth o vía USB. Este software es un producto de código abierto libre y está sujeto a la Licencia Pública General GNU (General Public License). El toolbox fue desarrollado en la Universidad de Aachen RWTH (Alemania), para estudiantes de ingeniería eléctrica, con una base en conocimientos básicos en Matemáticas, programación en C e Ingeniería de control.

Los estudiantes podrán realizar las mediciones de la señal y analizar las características de los diferentes actuadores y sensores de LEGO Mindstorms NXT desde el PC, así como implementar un sistema de control para una maqueta. El toolbox está diseñado principalmente para fines educativos.

El toolbox RWTH ofrece funcionalidad para los siguientes elementos:

•Conexión USB-Bluetooth.

•Los sensores del NXT (como por ejemplo: tacto, sonido, luz, ultrasonidos, brújula).

•Los servomotores NXT y funciones adicionales del NXT (como son: obtener el nivel de la batería, un tono).

El toolbox RWTH para Matlab se hace idóneo para controlar un NXT desde un ordenador por las siguientes características:

•Precisión y rapidez en los movimientos del motor, con un grado de precisión ± 1%, gracias a la tecnología híbrida de Motor Control.

•Fácil de utilizar los comandos de alto nivel, como la comunicación PC-NXT. No se requiere el conocimiento del protocolo de comunicación de PC-NXT.

•Documentación completa, con descripciones y ejemplos para cada función.

•Incorpora funciones que optimizan el rendimiento para obtener el mejor resultado posible.

•Plataforma Independiente, se puede instalar en Windows, Linux y Mac OS.

•Permite combinar aplicaciones a un NXT con operaciones matemáticas complejas y visualizaciones en Matlab.

En comparación con otros lenguajes de programación este presenta las siguientes ventajas.

•Enorme aumento de potencia de la CPU y la memoria disponible.

•El tamaño del programa es mayor que otros programas

•Control de múltiples robots con un solo programa (sólo limitado por el hardware).

•Se ejecuta el programa desde Matlab mientras que los programas clásicos NXT se ejecutan en el NXT.

•El uso de hardware adicional como por ejemplo, joysticks, etc.

•Características avanzadas de depuración, como puntos de interrupción, la ejecución paso a paso sobre la marcha e inspección de variables.

•Guía de uso del toolbox en Matlab para desarrollar aplicaciones.

4.4.2 Requisitos.

•Requisitos necesarios:

•Sistema operativo: Windows, Linux o Mac OS.

•MATLAB, versión 7.7 (R2008b) o superior.

•LEGO ® Mindstorms NXT kit de construcción.

•LEGO ® Mindstorms NXT firmware v1.26 o compatible.

•Bluetooth:

- Adaptador Bluetooth. Se recomienda AVM BlueFRITZ! USB compatible con el perfil de puerto serie (SPP).

- MINDSTORMS NXT Driver "Fantom", v1.02 o superior

•USB:

- MINDSTORMS NXT Driver "Fantom", v1.02 o superior

•Información adicional.

•Bluetooth:

No todos los adaptadores bluetooth son válidos y compatibles. Se debe utilizar el recomendado por RWTH, el adaptador AVM BlueFRIZ!.

•USB:

Para las conexiones USB en Windows, se necesita el driver fantom v1.13 [10].

•NeXTTool:

El programa NeXTTool.exe se encarga de generar una señal para controlar los motores del NXT. Sin este programa las funciones de NXTmotor no funcionarían.
No se puede utilizar a la vez que RobotC ya que NeXTTool necesita ejecutar un programa en el NXT para su funcionamiento. Una vez instalado el software y comprobado su funcionamiento, puede dar un error en el motor control al ejecutar el programa debido a la poca batería del NXT.

• MINDSTORMS NXT Firmware:

Debe de estar instalado el firmware 2.0 como mínimo para trabajar con el programa Motor Control.

4.4.3 Instalación paso a paso.

Se descarga el toolbox desde la página oficial de RWTH [3]. Una vez descargado y descomprimido, se ubica el archivo donde uno lo desee, aunque se recomienda guardarlo en la carpeta de toolbox de Matlab en la unidad C. Para poder utilizar el Toolbox sin tener que estar en ese directorio, se añadirá a Matlab. Los pasos a seguir son:

- En la ventana principal de Matlab, ir a "File", "Set Path",”Add folder". Seleccionar la carpeta RWTHMindstormsNXT. Una vez hecho, pulsar aceptar y save.

Se debe realizar el mismo proceso para las carpetas tools y demos situadas en el subdirectorio de la carpeta RWTHMindstormsNXT.

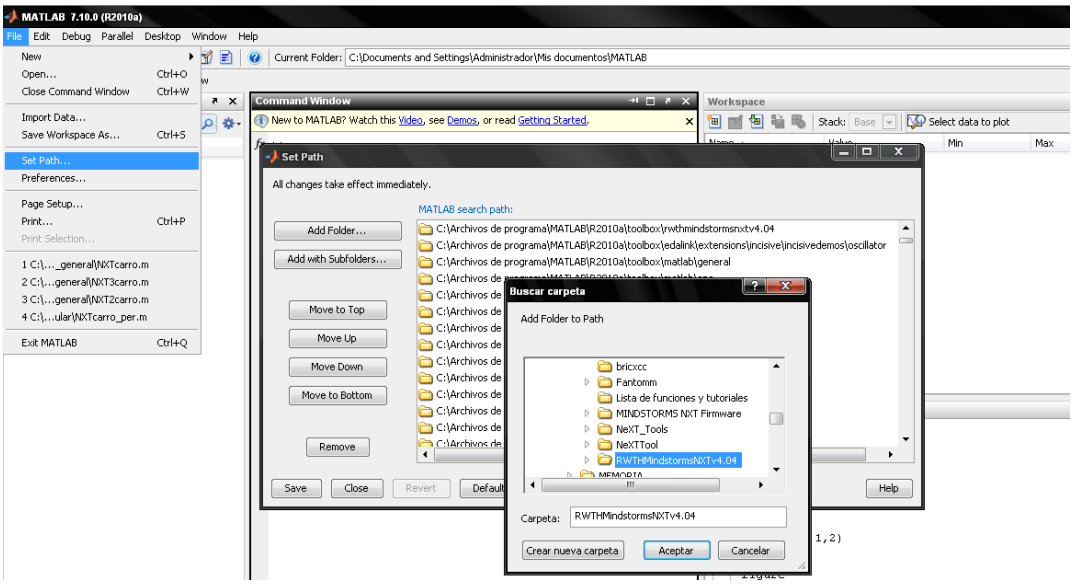


Figura 4.21. Instalación del toolbox.

Una vez realizado, ya se podrían utilizar los distintos comandos del toolbox para el NXT sin problemas.

4.5 GUI CONSTRUIDA

El objetivo fundamental de la construcción de la GUI para el proyecto es interactuar con la maqueta sin necesidad de conocimientos de informática previos facilitando así el manejo de esta.

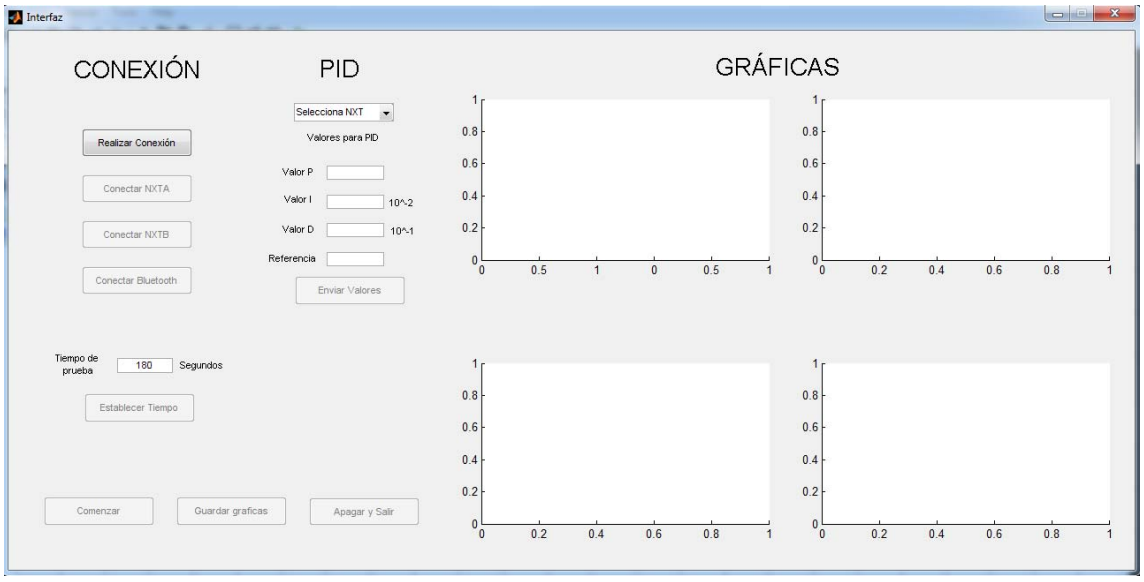


Figura 4.22 Interfaz GUI

Como se observa en la figura 4.22. La GUI consta de varias partes:

En la primera parte, la GUI permite realizar las conexiones entre los diferentes dispositivos de la maqueta gracias a una serie de botones encontrados en la parte izquierda superior.

Describiendo más detalladamente la función de cada botón:

-Realizar Conexión: Muestra en pantalla un mensaje de texto preguntando si se quieren realizar las conexiones. Si se selecciona la respuesta sí, se habilitan los botones Conectar NXTA y Conectar NXTB.

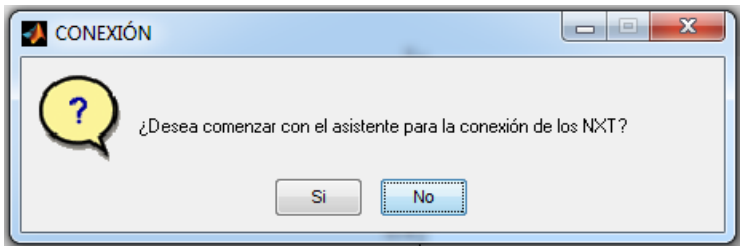


Figura 4.23. Mensaje de texto del botón Realizar conexión.

-Conectar NXTA: Conectar el brick NXTA a la GUI para poder utilizarlo.
Si la conexión se ha realizado correctamente, se habilita el botón Conectar Bluetooth.

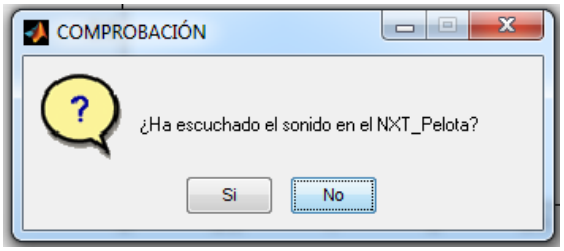


Figura 4.24. Mensaje de texto del botón Conectar NXTA al inicio.

-Conectar NXTB: Al igual que el botón anterior, pero conecta el brick NXTB.

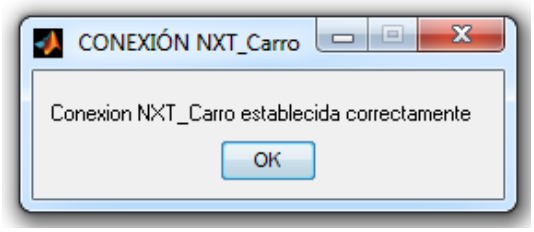


Figura 4.25.Mensaje de texto el botón Conectar NXTB al finalizar.

-Conectar Bluetooth: Conecta los dos brick por bluetooth mediante la ejecución de un programa llamado cbluetooth.rxe, incluido en el brick NXTA. Su código está contenido en el Anexo A.

Una vez Conectado, se habilita el botón Establecer tiempo.

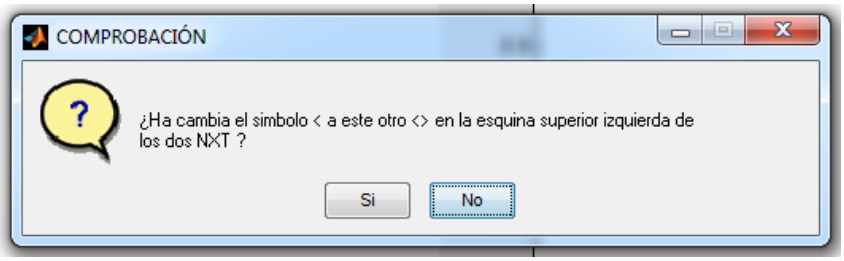


Figura 4.26. Mensaje de texto del botón Conectar Bluetooth.

Otra parte de la GUI permite enviar los valores del PID a los dos NXT, aunque como explicaremos en el Anexo C1, solo es posible realizar el envío al NXTA que controla la pelota. En esta parte se debe introducir cada uno de los valores del PID en su respectivo cuadro de texto. Para el caso del parámetro I, el valor será multiplicado por 0.01 y para el caso del parámetro D, el valor será multiplicado por 0.1.
Se puede encontrar una explicación más detallada en el Anexo C2.

Una vez seleccionado en el menú el NXT al que se le va a realizar el envío y escritos los parámetros del PID, solo falta pulsar el botón enviar.

Al inicio mostrará el valor del PID que tiene la pelota antes de comenzar.

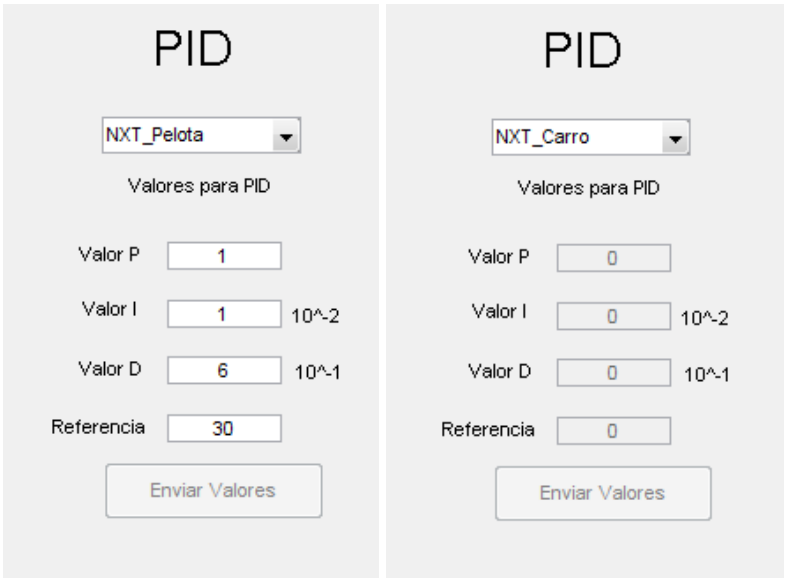


Figura 4.27. Parte GUI para enviar PID.

La parte más visible de la que consta la GUI son las gráficas (axes). Estas graficas muestran en tiempo real la medida de los sensores que se utilizan en la maqueta. Como se muestra en la figura 4.28, la GUI contiene cuatro gráficas:

- La primera grafica muestra la altura de la bola medida con el sensor de ultrasonidos en azul y a su vez la referencia que tiene que seguir la bola en rojo.

- La segunda grafica (superior derecha) muestra la altura a la que se encuentra el carro mediante el encoder de uno de los dos motores en azul. Se encuentra en rojo la referencia que tiene el carro que a su vez es la misma que la altura que tiene la bola en ese momento.

- La tercera gráfica muestra la tensión que tiene el ventilador que mueve la bola.

- La cuarta gráfica muestra la tensión que tiene uno de los dos motores a la hora de subir o bajar el carro por el tubo.

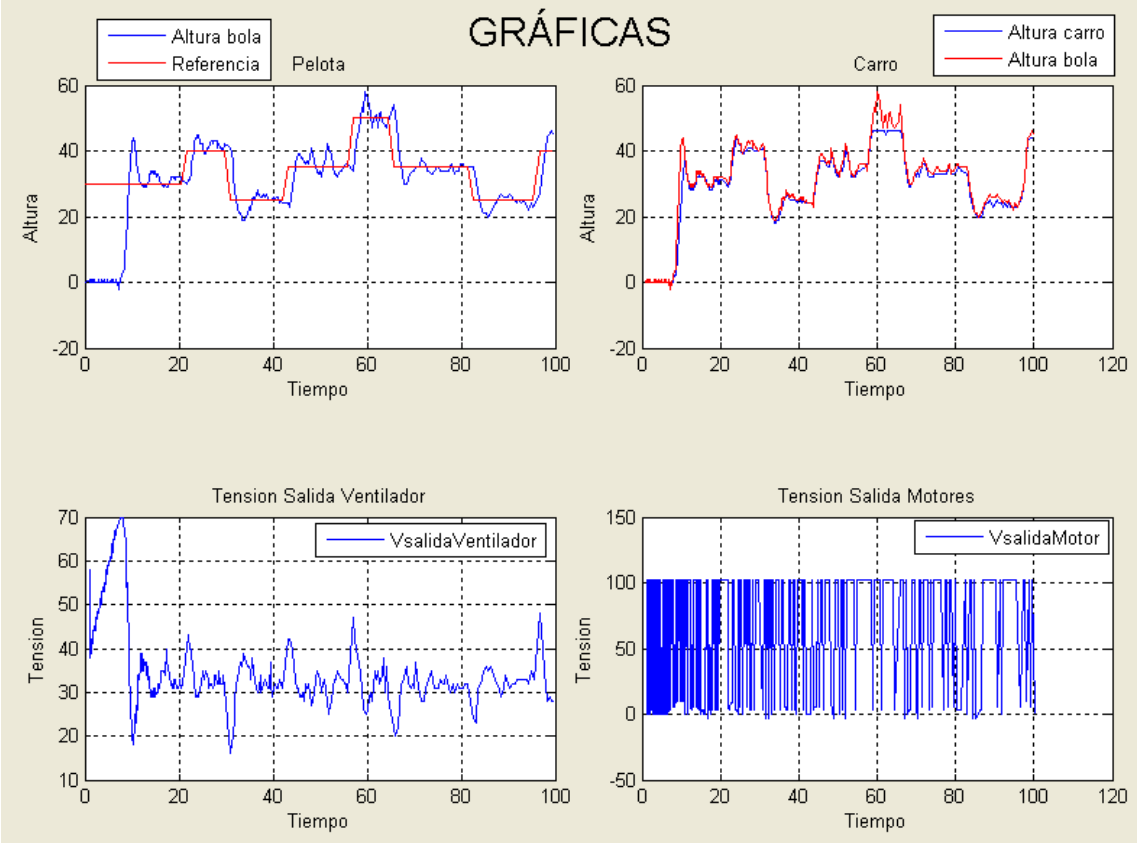


Figura 4.28. Graficas GUI.

La última parte que falta por comentar, es la parte inferior izquierda donde encontramos diferentes botones (Figura 4.29):

- Establecer tiempo: Sirve para definir el tiempo de duración en segundos de la maqueta para una prueba en concreto. El tiempo debe ser introducido en el cuadro de texto superior a este botón. Consta de una protección por si el texto que se introduce no es numérico. Una vez establecido el tiempo, se habilita el botón Comenzar.

- Comenzar: Este botón es de los más importantes de la GUI, ya que ejecuta los programas de RobotC en cada NXT y lee las medidas de los sensores para mostrarlas en las gráficas correspondientes. Al inicio habilita el botón Enviar valores y deshabilita los

botones Guardar gráficas y Apagar y salir. Borra los dibujos de las gráficas existentes y comienza a leer y dibujar las gráficas incluyendo su título y etiquetas de los ejes X e Y.

Una vez pasado el tiempo establecido, se incluye en las gráficas las leyendas, para solucionar el problema explicado en el Anexo C5. Se habilita el botón Guardar gráficas y Apagar y salir, se deshabilita el botón Enviar valores para evitar errores.

-Guardar gráficas: Este botón se habilita cuando ha acabado la prueba y no hay nada en ejecución. Copia y guarda cada una de las gráficas a un archivo .fig diferente

-Apagar y salir: Apaga los dos NXT y cierra la interfaz GUI.

Para apagar los dos NXT se ejecuta un programa en cada NXT llamado Apagar.rxe. El código de este programa está incluido en el Anexo A.

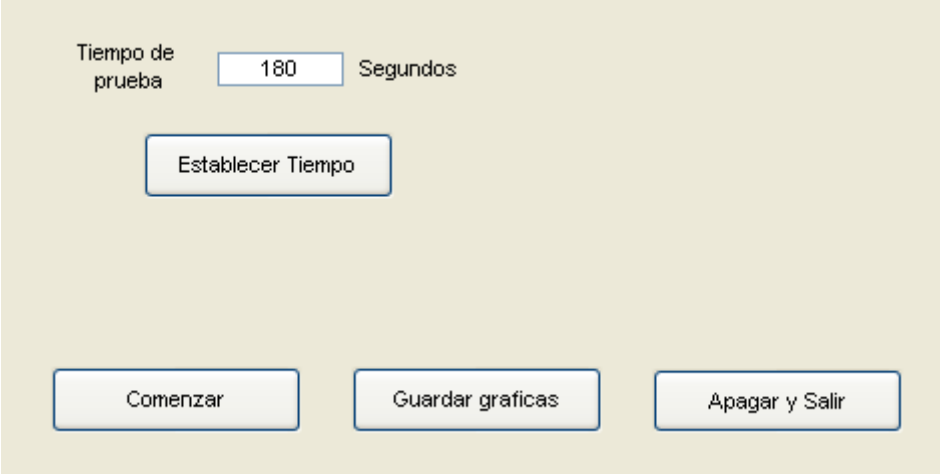


Figura 4.29.Seccion inferior izquierda GUI.

Todos los códigos desarrollados en la GUI se encuentran en el Anexo B.

En conclusión, la GUI al finalizar una prueba debe de quedar como la figura 4.30.

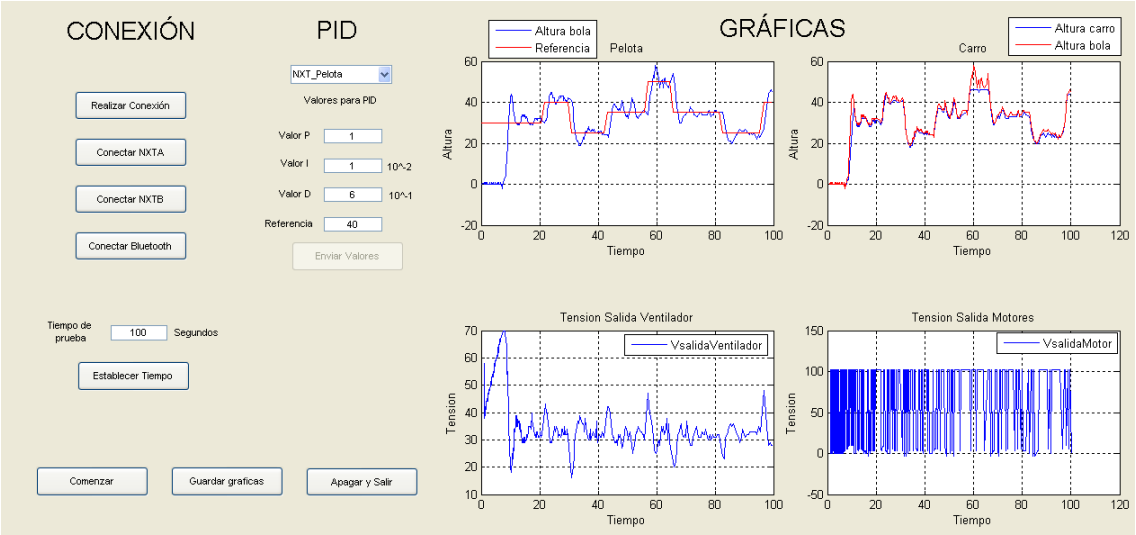


Figura 4.30. GUI al finalizar una prueba.

CAPITULO 5: COMUNICACIÓN ENTRE DISPOSITIVOS DE LA MAQUETA

En este capítulo se va a explicar cómo se realizan las diferentes conexiones entre los dispositivos de la maqueta.

Este es un tema delicado ya que por defecto la pila Bluetooth que maneja Windows no es compatible para poder conectar al LEGO. El NXT utiliza Bluetooth standard, pero no admite por ejemplo el transfer protocol de archivos u otras características más avanzadas. El Lego NXT puede mantener conexiones entrantes con tres partners simultáneos, además de mantener un cuarto canal especial para los mensajes salientes. Los mensajes pueden ser: números, valores booleanos (true o false), o cadenas de texto.

5.1 Conexiones de los diferentes dispositivos.

5.1.1 Entre PC y NXT.

Para este proceso, se debe utilizar el toolbox RWTH. Se divide en dos apartados según la tecnología utilizada:

• Bluetooth.

Para el caso de realizar una conexión por bluetooth en Matlab, se debe crear previamente un archivo de configuración de esta conexión.

Dentro de Matlab, se ejecuta la siguiente Instrucción: COM_MakeBTConfigFile
Una vez hecho, aparecerá la ventana de la figura 5.1.

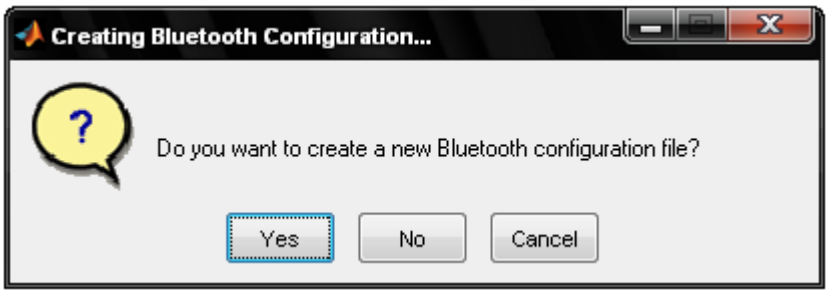


Figura 5.1. Ventana configuración bluetooth.

Dando clic en el botón “Yes”, se abrirá una nueva ventana, en donde se debe indicar la localización de la configuración del Bluetooth. Cuando se especifique la localización, aparecerá una nueva ventana. Esta ventana (figura 5.2) muestra los parámetros necesarios para realizar la conexión a Bluetooth. El parámetro que se debe modificar, es el que indica el puerto por el cual se establecerá la conexión. En este campo se pondrá el puerto que indica en el programa de BlueFRITZ!,

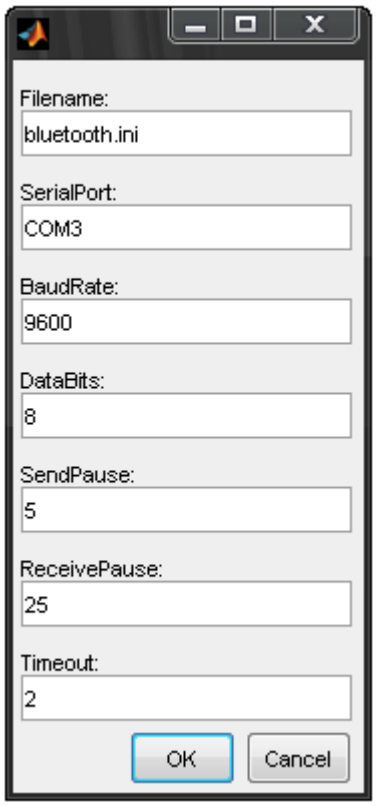


Figura 5.2 Parámetros bluetooth.

Una vez creado este archivo, ya se puede establecer la conexión entre PC y NXT utilizando el asistente de BlueFRITZ!

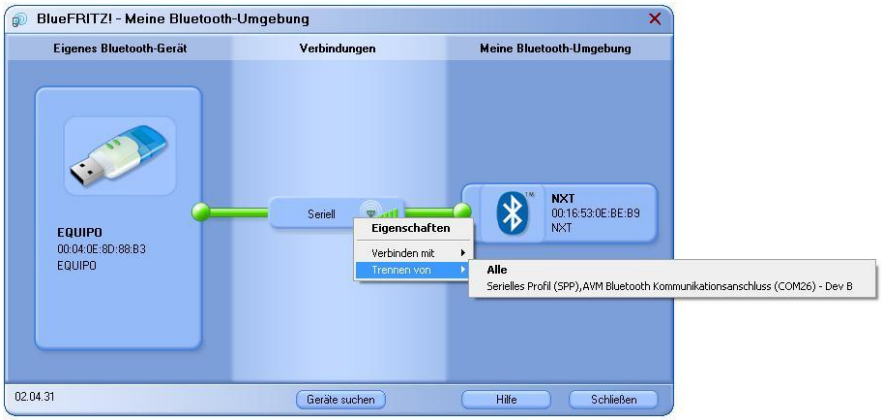


Figura 5.3. Establecer conexión.

Una vez establecida, se deben ejecutar las siguientes líneas.

```
COM_CloseNXT all%Elimina cualquier conexion
close all
clear all
hNXT = COM_OpenNXT('bluetooth,ini);
```

Donde Bluetooth.ini es el archivo de configuración para esta conexión.

• USB.

En Matlab introducimos los siguientes comandos:

```
COM_CloseNXT all%Elimina cualquier conexion
close all
clear all
hNXT = COM_OpenNXT();
COM_SetDefaultNXT(hNXT); %Toma por defecto este NXT
```

Si se desea conectar el NXT mediante su MAC,se debe sustituir la línea:
```
hNXT = COM_OpenNXT();
```

por este otro commando:
```
hNXT=COM_OpenNXTEX('USB','MAC');
```

5.1.2 Entre NXTA y NXTB.

Para conectar un NXT con otro, se debe activar el bluetooth desde el menú, en los dos NXT. Una vez activado, en ese mismo menú aparece la opción seach. Esta opción permite buscar al otro NXT mediante bluetooth. Una vez encontrado el NXT deseado, se selecciona y se conectan automáticamente. Si se desea realizar la conexión desde RobotC, se utilizan los siguientes comandos:

```
task main()
{
	setBluetoothOn();//Enciende el bluetooth
	wait10Msec(250);
	btConnect(2, "NXTB");//Se intenta conectar al bluetooth llamado NXTB
	wait10Msec(20);


}
```

5.2 Envío y recepción de datos entre dispositivos.

5.2.1 Envío de PC a NXT

En Matlab para enviar un mensaje al NXT después de haber establecido la conexión, se usa el siguiente comando:

NXT_MessageWrite(mensaje)

Este comando envía el valor que hay entre paréntesis a la posición 0 del buzón del NXT (mailbox).

5.2.2 Lectura del NXT desde PC.

• Lectura de sensores.

Para leer un sensor desde Matlab se debe abrir el puerto que se va a utilizar y luego leer el valor deseado. Como cada sensor tiene un comando diferente, se debe abrir un puerto con la función que empiece por open y contenta el nombre del sensor. Por ejemplo para utilizar un sensor de ultrasonidos en el puerto 3 el comando sería:

OpenUltrasonic(SENSOR_3)

Cuando ya se haya abierto el puerto en el que está conectado el sensor, se debe utilizar la función que comienza por Get y contenga el nombre del sensor. En este ejemplo, se utiliza la función: GetUltrasonic(SENSOR_3)

• Lectura de variables.

5.2.3 Envío y recepción de datos en el NXT.

• Envío de datos

Para enviar datos de un NXT a otro NXT, una vez establecida la conexión, se escribe en RobotC:

```
int x1 , x2 , x3;
x2 =100;
x3 =200;
x1 = 540;
sendMessageWithParm (x1 ,x2 ,x3 );
wait1Msec (100);
```

El comando sendMessageWithParm (x1 ,x2 ,x3),permite enviar hasta 3 valores al otro NXT.

• Recepción de datos

En RobotC se utilizan los siguientes comandos:

bQueuedMsgAvailable () :es una función que da un valor booleano, es verdadero cuando hay un mensaje en cola y falso en caso contrario.

messageParm [ ] :Lee el mensaje de la cola. De las tres casillas que existen y dependiendo de cuál se quiera leer, se pone 0, 1 ó 2 entre los corchetes.

Si el valor que se intenta leer ha sido enviado desde PC, solo se podrá utilizar el valor 0.

ClearMessage ( ); Elimina el mensaje actual de la cola.

• Leer sensores desde NXT en RobotC

Para leer los sensores desde RobotC, se deben declarar esos sensores en el programa. Dentro de RobotC en Robot, Motors and Sensor Setup, se definen los sensores que se van a utilizar y su nombre característico. Una vez hecho esto, aparece en la parte superior un código similar a este:

```
#pragma config(Sensor, S1,    sonar,           sensorSONAR)
#pragma config(Motor,  motorA,          ventilador,          tmotorNormal, openLoop, )
```

En este caso, se ha declarado un sensor de ultrasonidos en el puerto 1 y con un nombre característico llamado sonar. Para leer este sensor se debe escribir:

```
X=SensorValue[S1].
```

A la variable X se le asignará el valor que contenga ese puerto, concretamente el valor que proporciona ese sensor. Si se desea leer el valor que tiene el encoder de un motor, se utiliza la función:

```
X=nMotorEncoder[nombre_del_motor]
```

La variable X obtendrá el valor que tiene el encoder del motor en ese momento.

5.2.4 Combinación de RobotC y Matlab para leer variables

Un NXT no puede enviar valores directamente a PC, por lo que se tuvo que buscar una manera. El NXT tiene 4 puertos para conectar sensores. Si a un puerto libre (sin sensor conectado) se le proporciona un valor y se intenta leer desde Matlab, se obtendrá el valor asignado en el PC. En RobotC se escribe:

```
SensorValue[S4]=distancia;
```

Este comando asigna el valor de la variable distancia al puerto S4. Si con Matlab se intenta leer ese puerto, aunque realmente no hay conectado nada físicamente, se obtendrá ese valor.

Para leer un puerto desde Matlab se debe abrir el puerto y leer ese valor.

```
OpenUltrasonic(SENSOR_4,'SINGLE_SHOT',hNXTA); %Abre el Puerto 4

data=NXT_GetInputValues(SENSOR_4,  hNXTA); %Obtiene el valor de ese
puerto
b=data.ScaledVal; %Como no interesa todo, con este comando se obtiene
el valor deseado. En este caso el valor distancia.
```

5.3 Estudio de tiempos.

Para el estudio de tiempos, se realiza un pequeño programa con el que se obtiene la medida que genera el sensor de ultrasonidos.

5.3.1 Bluetooth.

Para ver los tiempos que tarda la conexión bluetooth entre PC y NXT se ha realizado la siguiente prueba en Matlab.

```
hNXTA=COM_OpenNXT('Bluetooth.ini');
OpenUltrasonic(SENSOR_1,'SINGLE_SHOT',hNXTA);
n=0;
a=0;
b=0;
x=0;
tic
while (n<25)
Getultrasonic(SENSOR_1,hNXTA)
n=n+1
x=[x n]
a=[a toc]
end
```

Se obtiene

a=[0 0.0511 0.0619 0.0821 0.1009 0.1199 0.1399 0.1599 0.1819
0.1999 0.2201 0.2399 0.2599 0.2799 0.2999 0.3199 0.3399 0.3611
0.3811 0.4021 0.4221 0.4421 0.4614 0.4821 0.5014 0.5212]

Con el valor de a obtenido, se utiliza el siguiente código:

```
n=1;
while n<24
b=[b (a(1,n+1)-a(1,n))]
n=n+1
end
```

La variable b que se obtiene es la velocidad de envío. Se puede ver mejor en la figura 5.4.

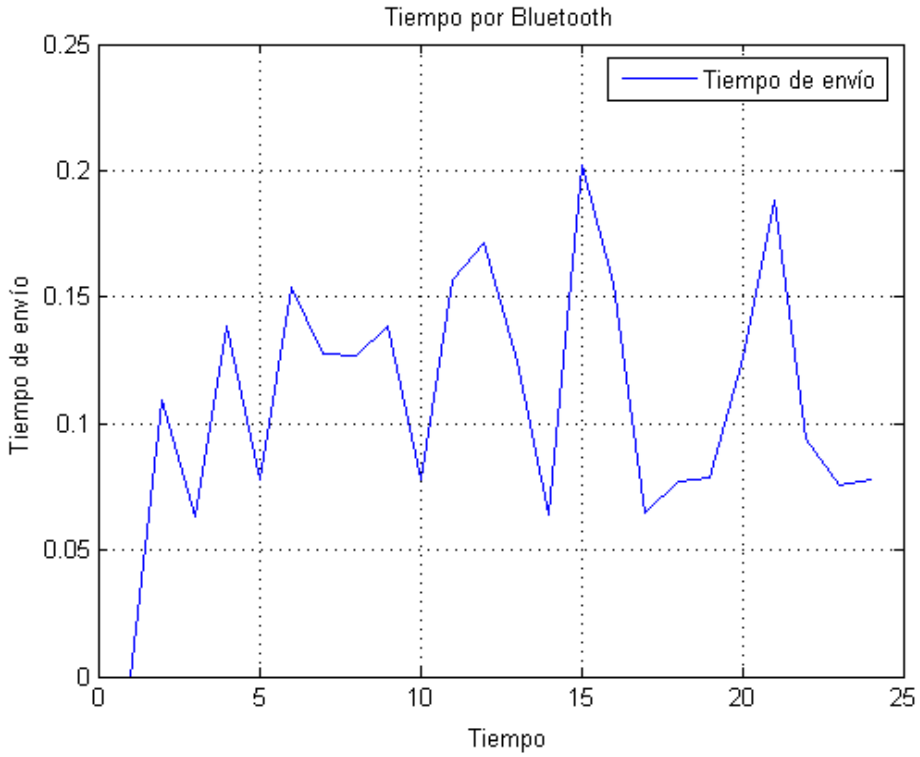


Figura 5.4. Tiempo de envío por Bluetooth.

Como se puede apreciar, el tiempo varía entre 0,08 y 0,2 segundos.

5.3.2 USB.

Este apartado se realiza casi igual que el apartado anterior.
En Matlab se debe escribir:

```
hNXTA=COM_OpenNXT();
OpenUltrasonic(SENSOR_1,'SINGLE_SHOT',hNXTA);
n=0;
a=0;
b=0;
x=0;
tic
while (n<25)
Getultrasonic(SENSOR_1,hNXTA)
n=n+1
x=[x n]
a=[a toc]
end
```

Se obtiene:

a=[0 0.0511 0.0108 0.0202 0.0188 0.0190 0.0200 0.0200 0.0220 0.0180 0.0202 0.0199 0.0200 0.0200 0.0200 0.0200 0.0200 0.0212 0.0200 0.0210 0.0200 0.0200 0.0193 0.0207]

Con el valor obtenido se utiliza el siguiente código:

```
n=1;
while n<24
b=[b (a(1,n+1)-a(1,n))];
n=n+1;
end
```

Con el valor de b se obtiene la gráfica de la figura 5.5.

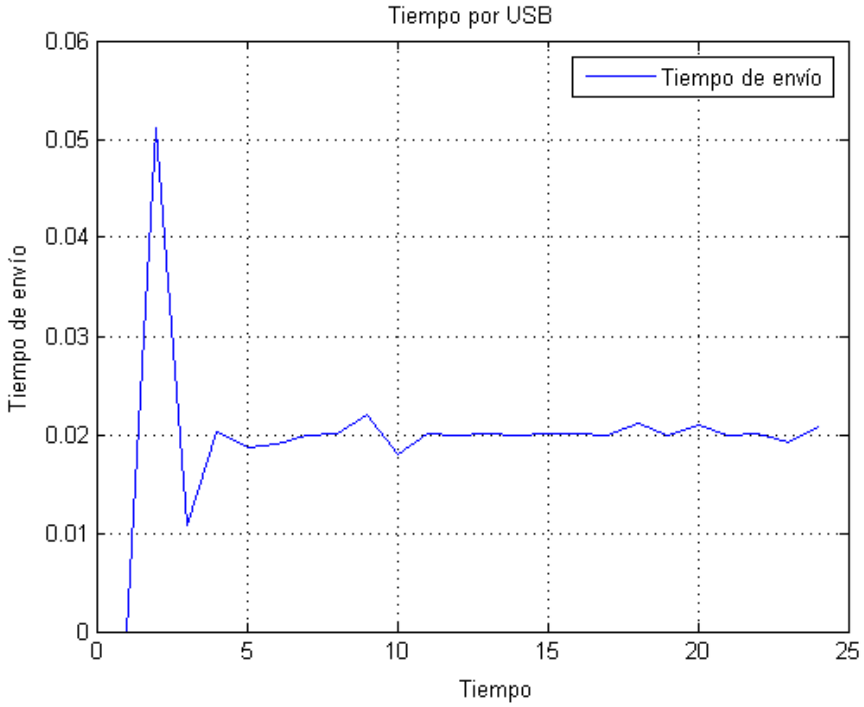


Figura 5.5. Tiempo de envío por USB.

Se observa que tiene un valor medio de 0.02 segundos. Como este método es más rápido que por bluetooth, se utilizó esta conexión para la maqueta.

5.3.3 RobotC.

Utilizando el siguiente código en RobotC:

```
int n=0;
TFileIOResult resultado; //Flag que nos indica el resultado de la operación con un archivo
TFileHandle etiqueta; //Etiqueta del archivo
int longitud = 25000; //Longitud de cada archivo en bytes
string nombrearchivo; //String que sera el nombre de cada archivo
task main ()
{
        nombrearchivo="tiempos.txt";
 string datosalida;
```

```
  Delete(nombrearchivo, resultado);
  OpenWrite(etiqueta, resultado,nombrearchivo, longitud);
        ClearTimer(T1);
  time1[T1]=0;
   while (n<50)
    {
        n=n+1;
datosalida=time1[T1];
sendMessageWithParm(1,1,1);
        WriteText(etiqueta,resultado,datosalida+"\t");
        wait10Msec(1);
  }
CloseAllHandles(resultado);
}
```

Se obtiene un valor de

a=[0	11	21	31	41	51	61	71	81	97	107	117
127	137	147	157	167	177	187	197	207	217	227	237
247	257	267	277	287	297	307	317	327	337	347	357
367	377	387	397	407	417	427	437	447	457	467	477
487	497]										

En Matlab se ejecuta el siguiente código:

```
n=1;
while n<50
b=[b (a(1,n+1)-a(1,n))]
n=n+1
end
```

Se obtiene un valor de b mostrado en la figura 5.6.

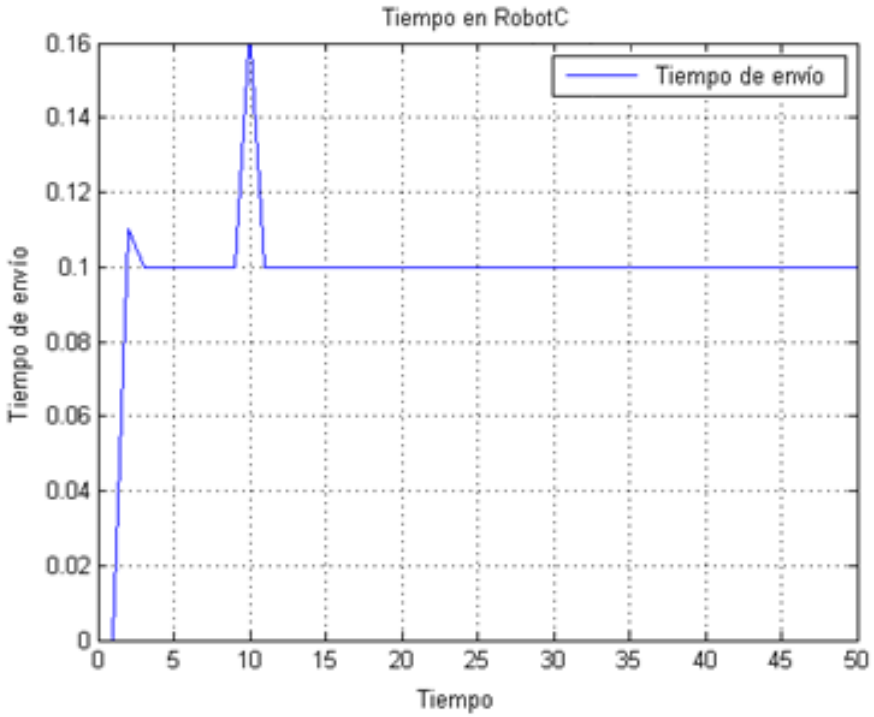


Figura 5.6. Tiempo de envío en RobotC mediante Bluetooth.

Se observa un tiempo de envío de 0,1 segundos y un pico de 0,16 segundos. Se debe tener en cuenta un wait, ya que sino la función de grabar texto en un archivo *.txt no funcionaría.

5.4 Problemas bluetooth.

El problema se debe al fallo de la conexión simultánea entre PC-NXTA y NXTA-NXTB. Realizando un programa en el que se envía un mensaje del PC al NXTA y seguidamente del NXTA al NXTB, se comprobó que no era posible realizarlo, ya que sin ningún motivo aparente, siempre se pierde una de las dos conexiones.
Estos dos envíos por separado se envían perfectamente por lo que no es un error de código, sino un error de conexión de bluetooth. Por lo que no es posible tener simultáneamente una conexión NXTA-PC y NXTA- NXTB. La figura 5.7 muestra la explicación del problema gráficamente.

Figura 5.7. Explicación gráfica del problema.

5.5 Solución de problemas.

En este apartado se va a explicar todas las ideas que surgieron para solucionar este problema y cuál fue la escogida.

5.5.1 Añadir un NXT.

La solución propuesta sería añadir un NXT más al proyecto para que este sea conectado por USB al PC y a su vez ejerza de maestro de los otros dos NXT que serían esclavos. Así solo se utiliza la conexión bluetooth entre los NXT por lo que no hay ningún problema de conexiones.

El inconveniente de esta solución es que las latencias entre NXT-NXT son más altas que PC-NXT por lo que tardaríamos más tiempo en enviar los valores por bluetooth y esto podría llevar a un funcionamiento del sistema más lento. Además se tendrían problemas para leer los valores de los sensores desde el NXT conectado por USB.

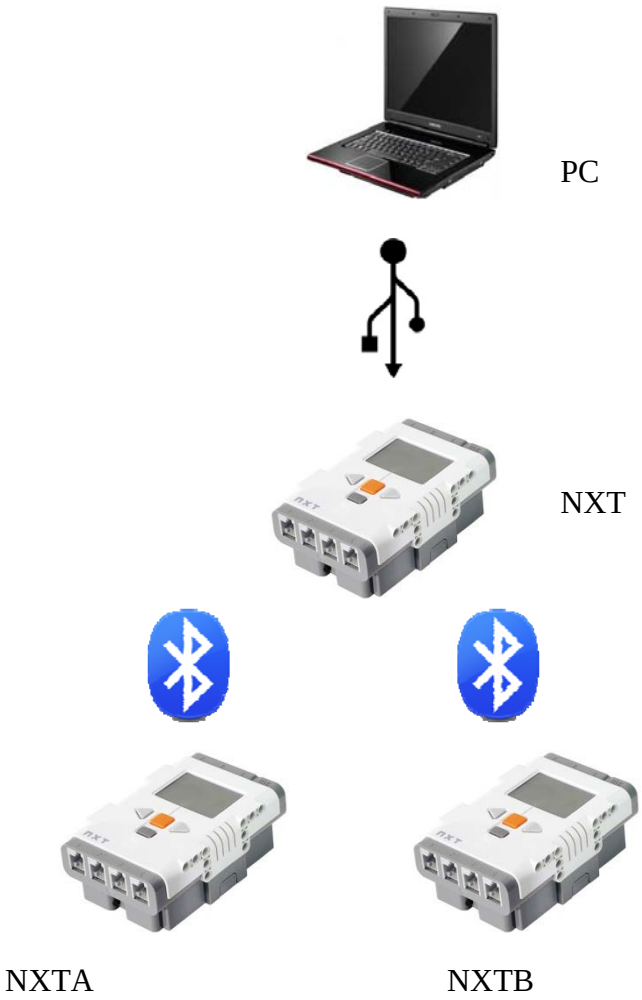


Figura 5.8. Solución con tres NXT.

5.5.2 Conexión conmutada entre PC-NXTA y PC-NXTB.

Otra solución sería ir conectando y desconectando el PC a los NXT que se necesitara, ya que así no se tendrían dos conexiones activas al mismo tiempo. El problema es que desde el toolbox de NXT de MATLAB no podemos establecer o desconectar estas dos conexiones como se quieran. Solo permite una vez conectado PC-NXT desde un asistente bluetooth, utilizar esa conexión. Para utilizar esta solución se debería de cambiar manualmente la conexión desde el asistente de bluetooth por lo que se descartó.

5.5.3 Conexión con dos bluetooth.

Si se añade otro bluetooth, es decir, el PC tendría dos bluetooth, se podrian conectar cada uno de ellos a un NXT y dejar esa conexión establecida mientras funciona la maqueta.

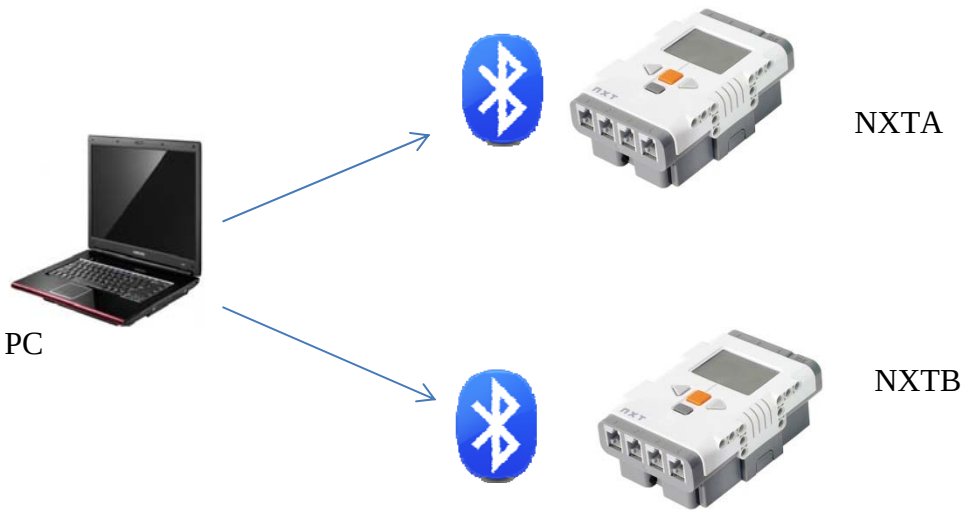


Figura 5.9. Solución con dos dispositivos bluetooth.

Esta solución no se llegó a poner en práctica ya que se pensó que no sería la mejor opción.

5.5.4 Utilizar solo un NXT.

Otra solución pensada para este problema sería sustituir los dos NXT por solo uno, que haga la misma función que los otros dos por separado. Este NXT controlaría a la vez la pelota y el carro con sus respectivos sensores y a su vez conectado por bluetooth al PC.
Esta solución no tendría problemas aparentemente con la conexión bluetooth, ya que solo existe una conexión. El inconveniente que se podría tener, sería comprobar si es suficientemente rápido para controlar la pelota y el carro a la vez, incluso mientras envía y recibe valores desde bluetooth.

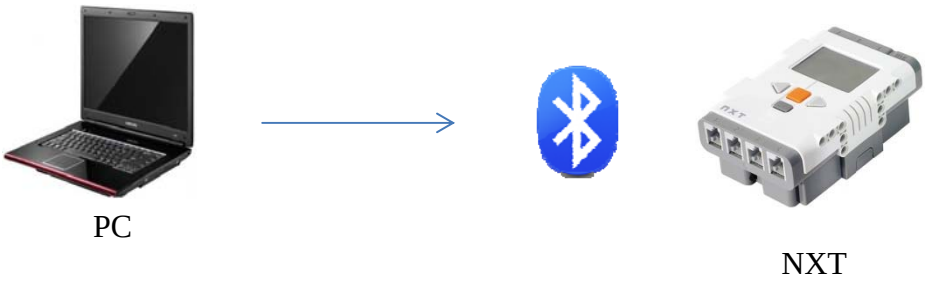


Figura 5.10. Solución con un solo NXT mediante bluetooth.

5.5.5 Conexiones de PC-NXTA y PC-NXTB por USB.

Esta solución es la utilizada, ya que solucionamos los tiempos y los problemas de conexión bluetooth.

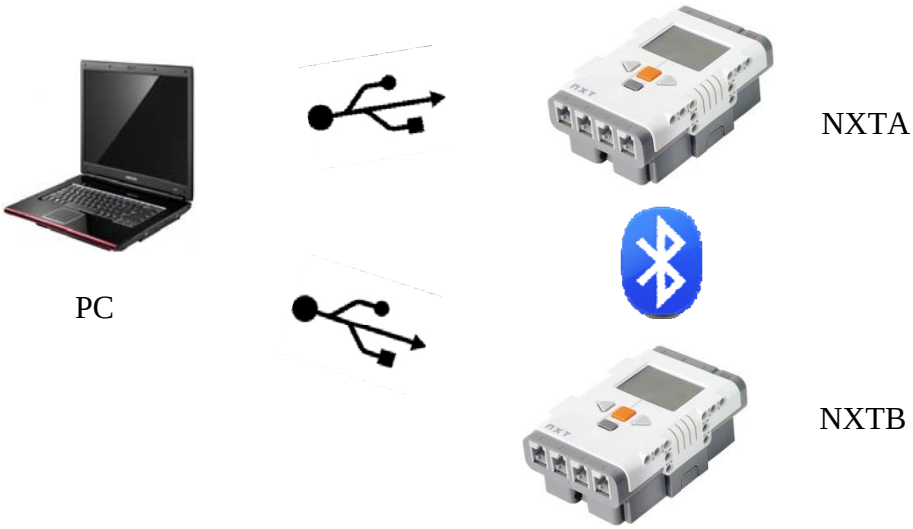


Figura 5.11. Solución final con conexión USB.

CAPITULO 6: IDENTIFICACION Y CONTROL

6.1 Identificación del modelo del prototipo que controla la Pelota.

Para la identificación de este modelo se realizaron diferentes experimentos.

6.1.1 Lazo abierto

Para intentar conseguir el modelo se realizó un experimento en lazo abierto dando un escalón de tensión al ventilador y midiendo la altura que sube la pelota.
No se pudo obtener de esta manera ya que el sistema no es lineal.

6.1.2 Lazo cerrado

Se realizó otro experimento en este caso en lazo cerrado con un proporcional de valor 2, donde se daba un escalón al sistema y una vez estabilizado, se le daba otro escalón en este caso de 20 unidades, con el que se va a trabajar.

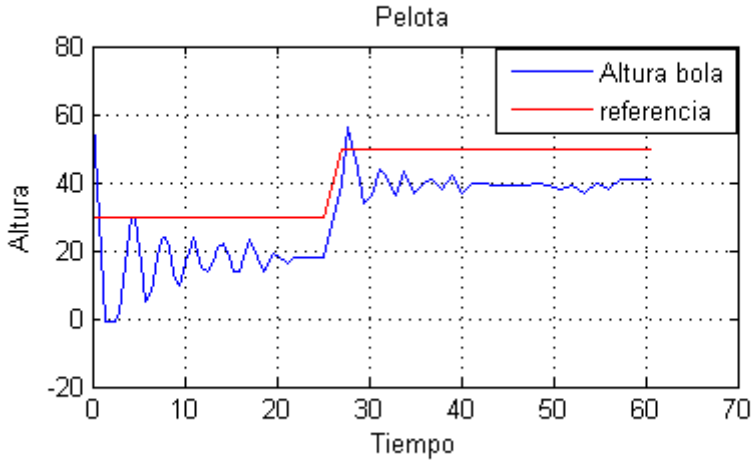


Figura 6.1. Salida en lazo cerrado.

Se pueden observar los dos escalones mencionados en la figura 6.1, aunque como se ha comentado anteriormente, solo se utiliza el escalón que comienza en el segundo 25.

Se van a identificar las dinámicas de este sistema como un sistema de primer orden y como un sistema de segundo orden.

•Como sistema de primer orden.

Tomando el modelo obtenido como un sistema de primer orden, se pueden calcular los siguientes valores:

Figura 6.2 Valor T

Tiempo de establecimiento: $T = 25.8 - 25 = 0.8$

Ganancia: $K = \frac{Y(S)}{U(S)} = \frac{22}{20} = 1.1$

Función de transferencia total: $M(S) = \frac{1.375}{S+1.25}$

Función de transferencia del sistema: $G(S) = \frac{M(S)}{1-M(S)}\frac{1}{P} = \frac{0.6875}{S-0.125}$

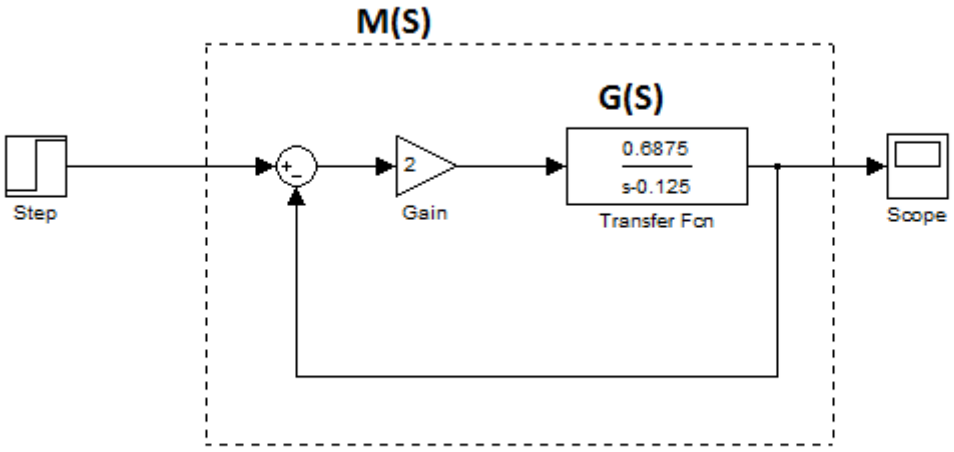


Figura 6.3.Sistema en simulink.

La figura 6.3 muestra la comparación del modelo aproximado obtenido con sistema real.

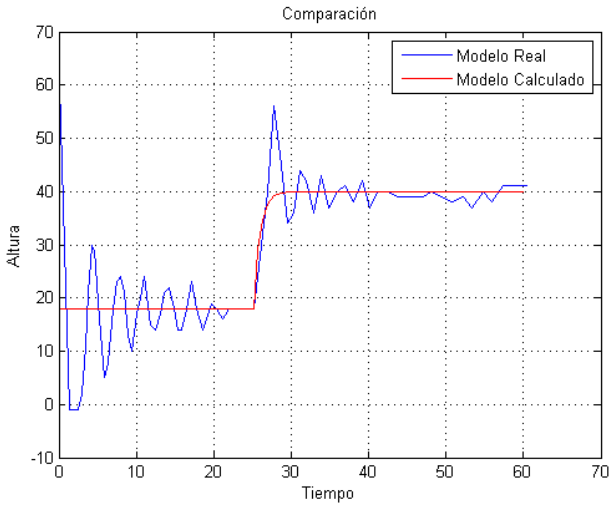


Figura 6.4. Comparación del sistema aproximado con el real.

Se puede observar en la figura 6.4 que este modelo no es el adecuado para trabajar con él.

•Como sistema de segundo orden.

Para calcular el modelo de segundo orden se deben calcular los siguientes valores:

Valor de pico: $Yp = 56 - 18 = 38\ Cm$

Valor en régimen permanente: $Yrp = 40 - 18 = 22\ Cm$

Tiempo de establecimiento: $Ts = 35 - 25.14 = 10.14\ s$

Ganancia: $K = \frac{Y(S)}{U(S)} = \frac{22}{20} = 1.1$

Sobrepico: $Mp = \frac{Yp - Yrp}{Yrp} = 0.72 = 72\%$

Coeficiente de amortiguamiento: $\xi = \text{Cos}\theta = 0.1$

$$\sigma = \frac{\Pi}{Ts} = 0.3$$

$$\theta = \frac{-\Pi}{Ln(Mp)} = 84º$$

Con todos estos datos se obtiene la función de transferencia:

Función de transferencia total:

Con esta función de transferencia se puede obtener G(S) sabiendo que P=2.

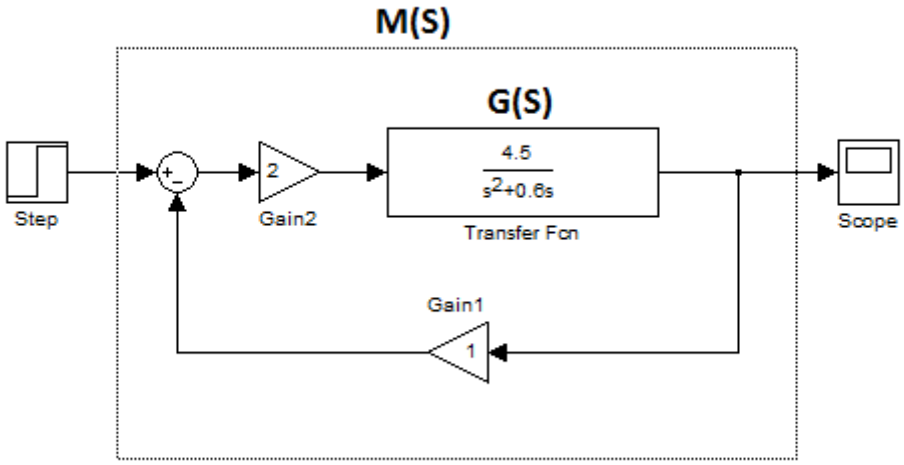


Figura 6.5.Sistema en simulink.

En la figura 6.6 se muestra la comparación del modelo calculado con el sistema real.

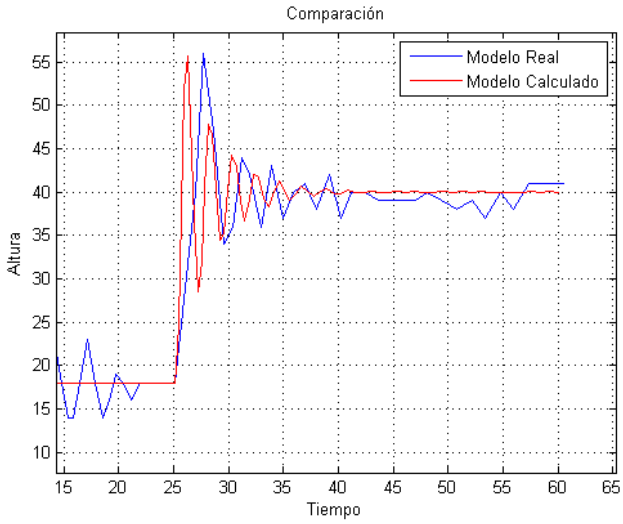


Figura 6.6. Comparación del modelo aproximado con el real.

Se observa que el modelo calculado es el adecuado para trabajar con él.

6.2 Control Pelota.

Para sintonizar el controlador de la altura de la pelota en el tubo, se han utilizado los dos métodos de Ziegler-Nichols.

6.2.1 Segundo Método de Ziegler-Nichols.

Este método se realiza en lazo cerrado, variando una ganancia, Kc, hasta llegar al límite de inestabilidad en el que la salida sea una onda senoidal.
Cuando se obtiene el valor de Kc y el valor del periodo de la onda de salida Tc, se utiliza una tabla para calcular los parámetros del PID que controla el ventilador.

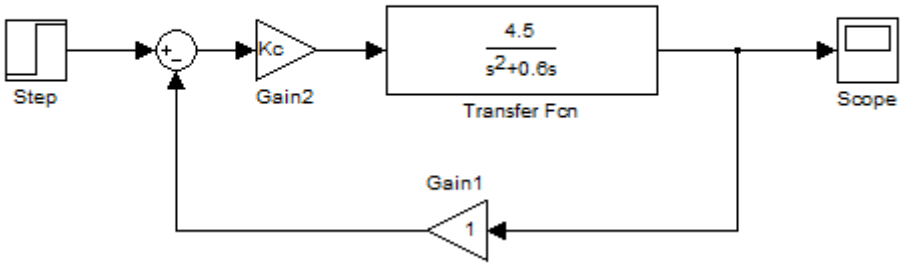


Figura 6.7. Sistema simulink.

Se observa en el lugar de las raíces de la figura 6.8 que G(S) tiene dos asíntotas ya que tiene dos polos y ningún cero.

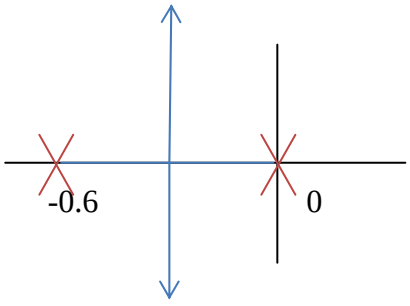


Figura 6.8. Lugar de las raíces.

Estas dos asíntotas provocan que el sistema no se inestabilice nunca cambiando el valor de Kc, por lo que se descarta este método.

6.2.2 Primer Método de Ziegler-Nichols.

Con este método se trabaja en lazo abierto. Para obtener un controlador adecuado, con una entrada de escalón unitario en lazo abierto, se debe hacer la tangente a la salida en el punto de inflexión máximo.Cuando esta tangente corta con los ejes de ordenadas y

abscisas, se obtienen los valores L, T y a. Utilizando la tabla de la figura 6.11, se obtienen los parámetros del PID que controla el ventilador.
Se van a utilizar L y a para la obtención de los parámetros.
El método para calcular los valores se muestra en la figura 6.10.

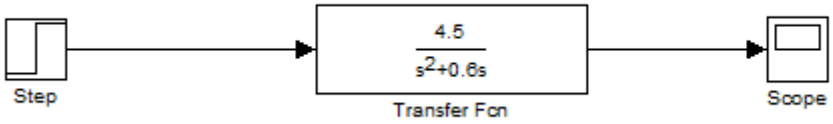
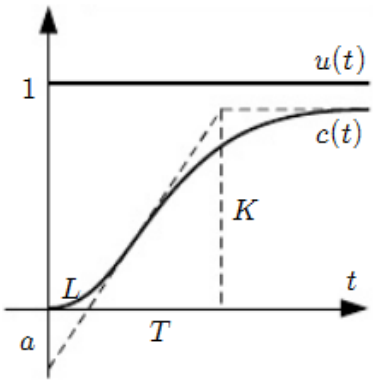


Figura 6.9. Sistema simulink.



$$a = \frac{KL}{T}$$

Figura 6.10. Valores gráficamente.

Tipo	K_p	T_i	T_d
P	$\frac{1}{a}$	∞	0
PI	$\frac{0.9}{a}$	$3L$	0
PID	$\frac{1.2}{a}$	$2L$	$0.5L$

Figura 6.11. Tabla de parámetros PID.

Trabajando con G(S) en lazo abierto se obtiene una salida mostrada en la figura 6.12.
Esta salida es imposible de tener en la maqueta, ya que el tubo no es lineal y nunca podría sobrepasar de 65 cm de altura.

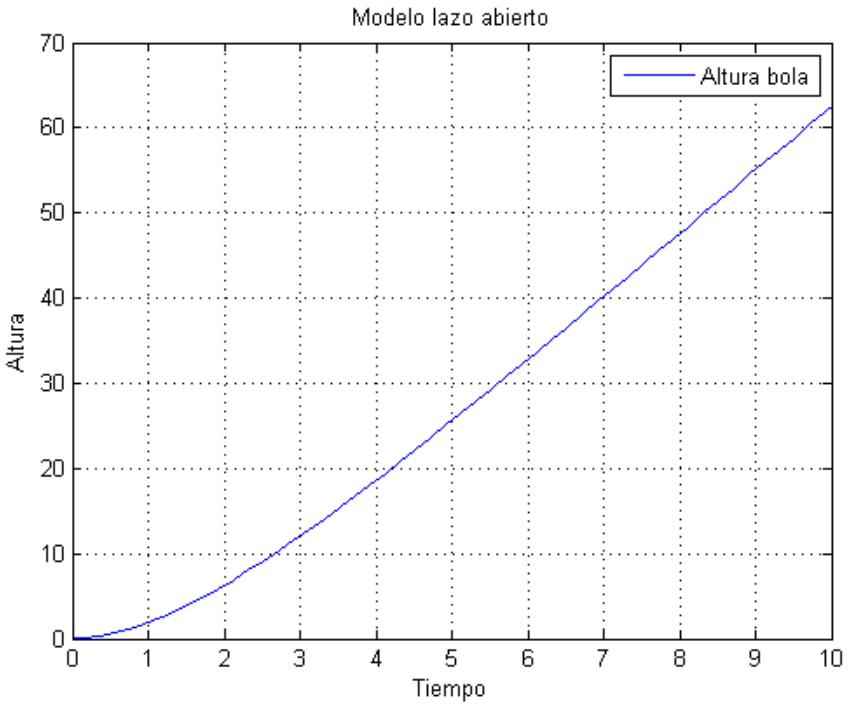


Figura 6.12. Salida lazo abierto.

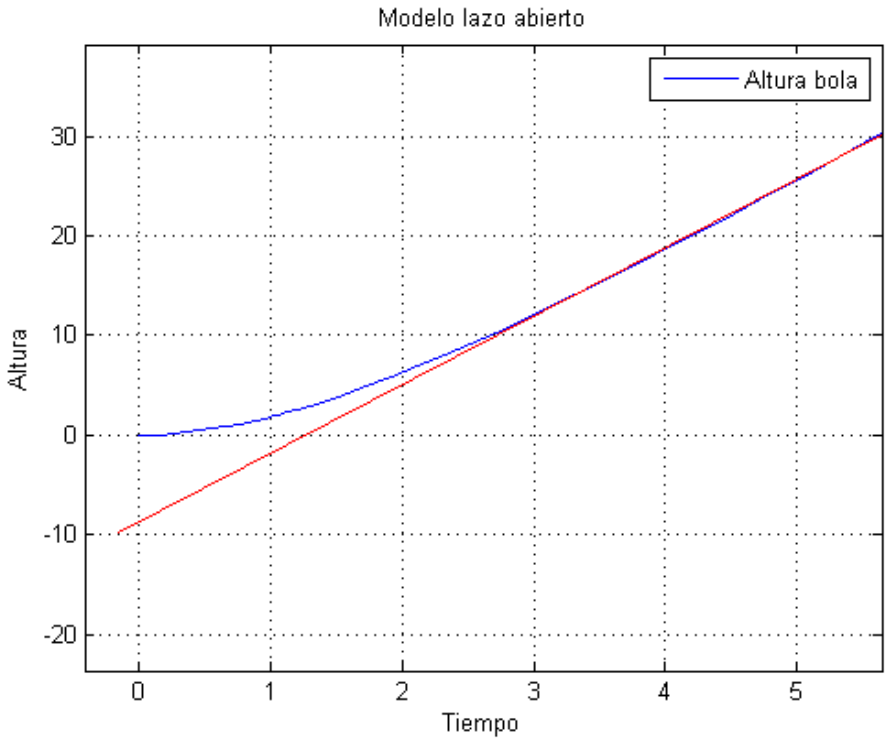


Figura 6.13. Tangente modelo lazo abierto.

Si se realiza la tangente en su punto de inflexión máximo como puede verse en la figura 6.13, se obtiene unos valores de: a=9 y L=1.2.

Para los valores obtenidos se pueden realizar tres controles distintos.

Para el control con un proporcional, se obtiene una $P = \frac{1}{a} = 0.1111$

Realizando el control en simulación con el modelo mencionado, se obtiene la siguiente salida:

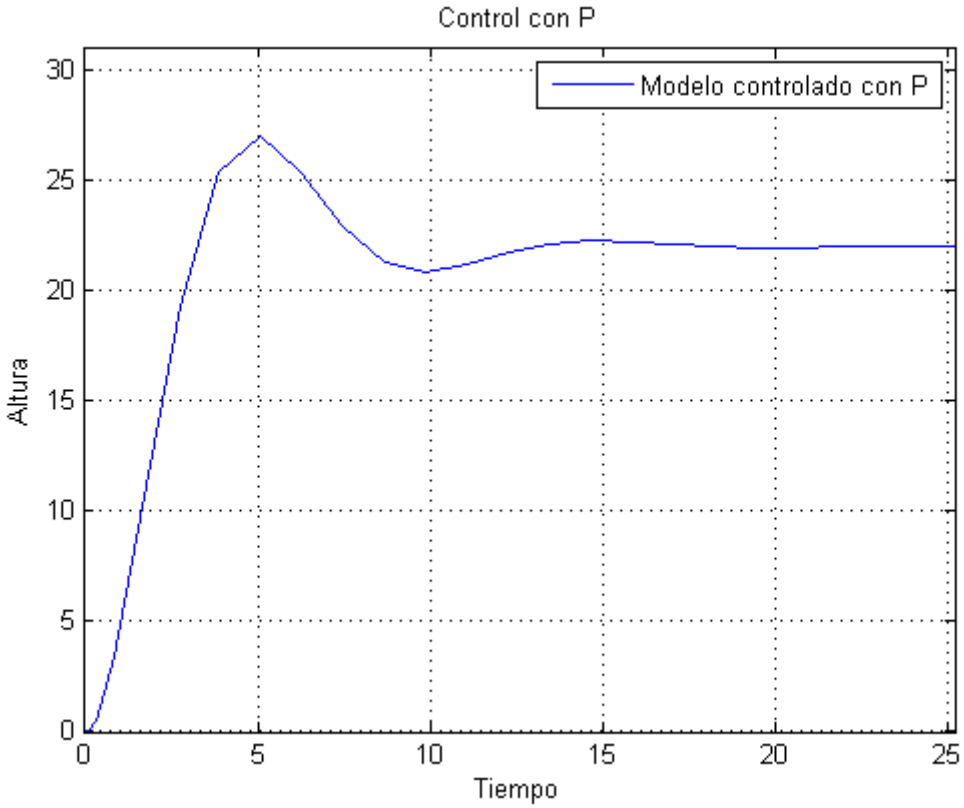


Figura 6.14. Salida con P.

Con un PI de $P = \frac{0.9}{a} = 0.1$; $I = \frac{K}{Ti} = \frac{0.1}{3L} = 0.0277$

Se obtiene la siguiente salida:

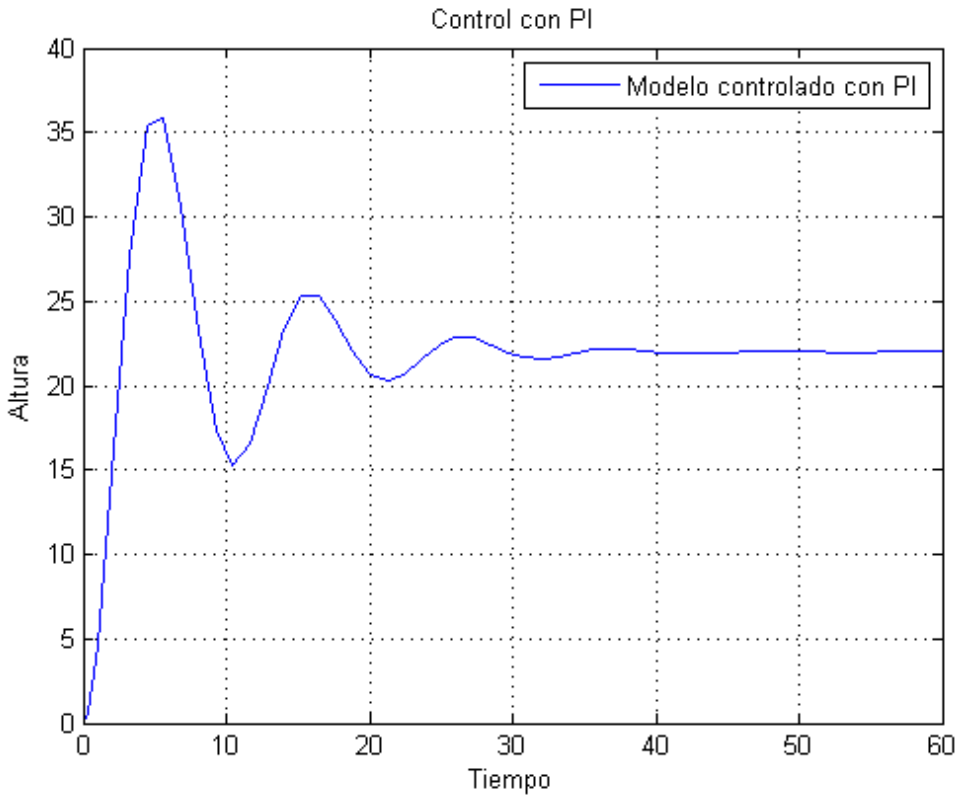


Figura 6.15. Salida con PI.

Los parámetros para un PID son:

$$P = \frac{1.2}{a} = 0.1333 \;;\; I = \frac{K}{Ti} = \frac{0.1333}{2L} = 0.0555 \;;\; D = K * Td = \frac{KL}{2} = 0.08$$

Se obtiene la siguiente salida:

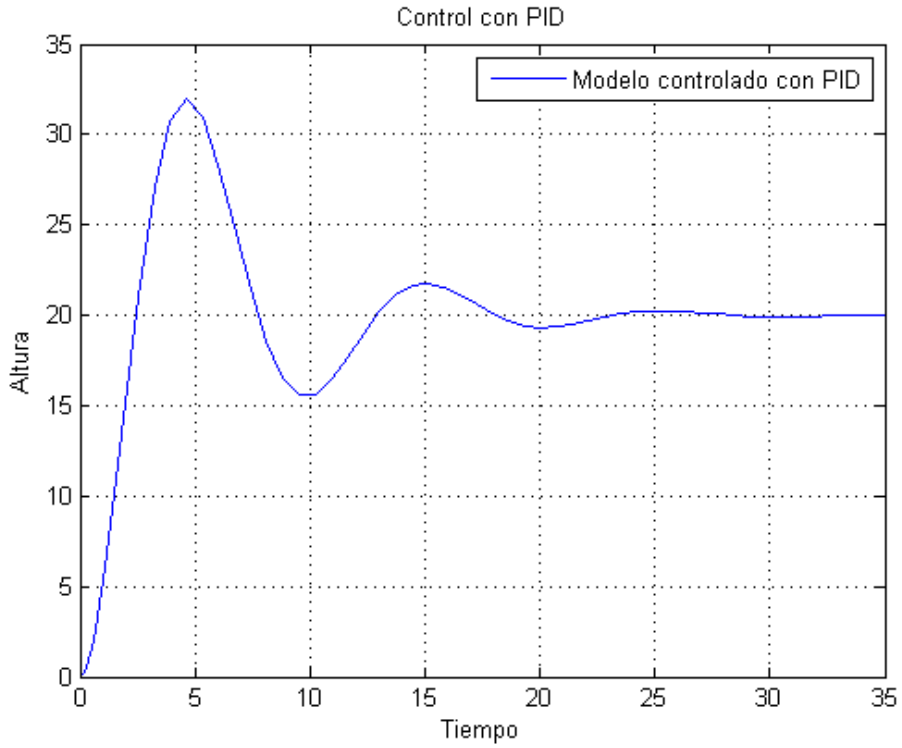


Figura 6.16. Salida con PID.

De los tres controles obtenidos, el controlador con un PID es el más adecuado.
Tiene menor sobrepico que los otros controladores y un tiempo de establecimiento menor. Por lo que se considera un control del sistema aceptable.

6.3 Control de la plataforma real.

Haciendo una prueba con los parámetros del PID obtenido, se comprobó que no funcionaba adecuadamente ya que el tubo no es lineal. Este control se ha realizado para una zona concreta del tubo.

Modificando los parámetros anteriores, se obtuvieron unos nuevos:
P=1.33
I=0.01
D=0.6

Estos parámetros funcionan adecuadamente como muestra la figura 6.17

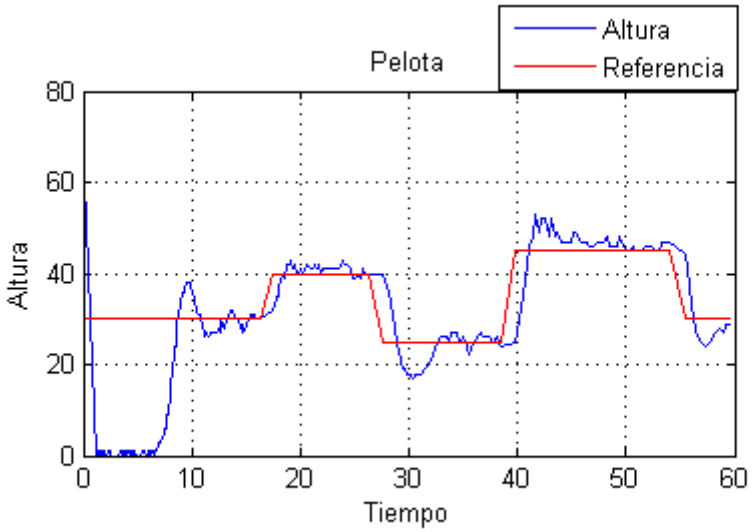


Figura 6.17 Rendimiento PID

A veces la pelota oscila en un rango de altura muy próximo a la referencia debido a que el sensor de ultrasonidos no lee bien la pelota cuando esta gira. Por lo que al cometer un error en la medida, el PID intenta corregirlo y se produce este pequeño inconveniente.

También hay que destacar que según la zona donde se encuentre la pelota, el PID funcionará mejor o peor, ya que la dinámica varía según la zona de funcionamiento por ser un sistema no lineal.

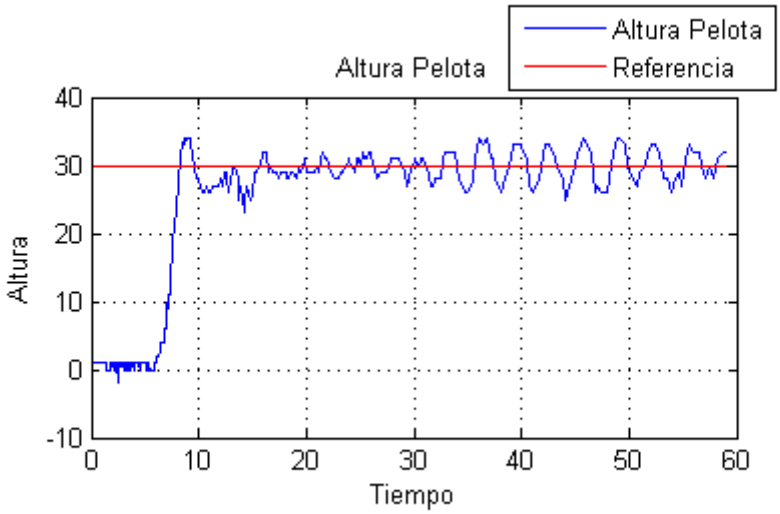


Figura 6.18 Altura con PID final.

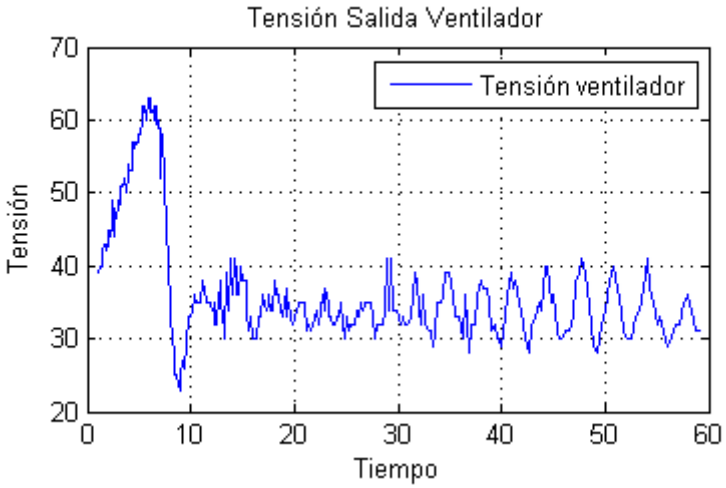


Figura 6.19. Tensión con PID final.

6.4 Identificación del modelo del prototipo que controla el Carro.

Para la identificación de este modelo se realizaron diferentes experimentos.

6.4.1 Lazo abierto.

El primer método que se optó para identificar el modelo del carro fue dar a los motores una tensión del 100% durante 2 segundos. El código utilizado es Tension_Motor.rxe incluido en el Anexo A. El inconveniente que surgió fue que la gráfica que se obtiene a la salida tenía forma de escalera por lo que incluso simplificándola como una recta, no se podía apreciar bien y llevaría a cometer demasiados errores.

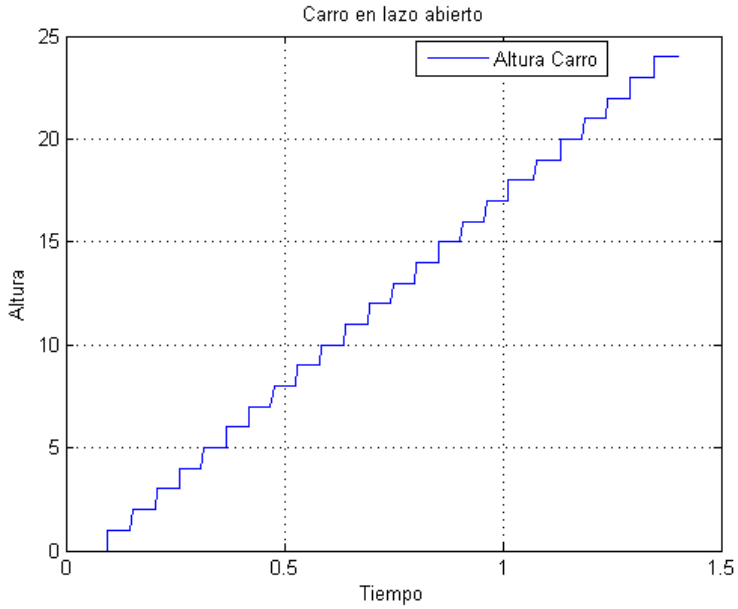


Figura 6.20. Salida lazo abierto.

Mejorando la estructura del carro, se obtuvo una salida mejorada como la de la figura 6.21

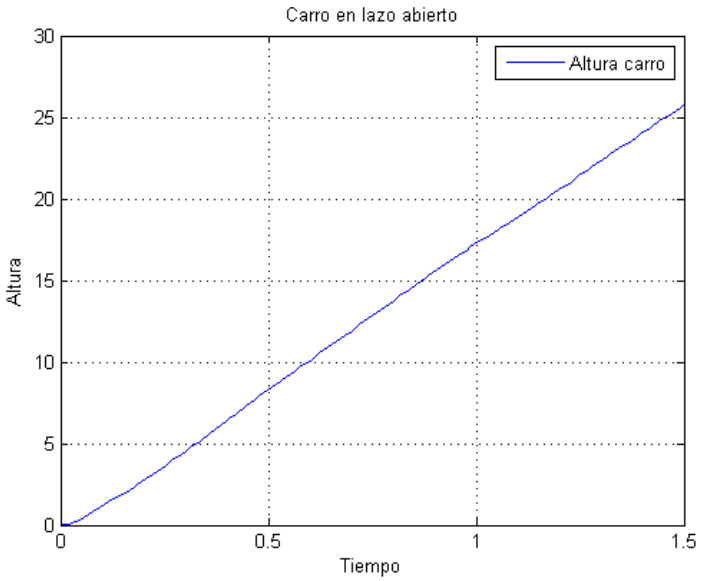


Figura 6.21.Salida carro lazo abierto final.

6.4.2 Velocidad.

Para obtener el modelo de la figura 6.23 se tuvo que calcular la velocidad. Una vez obtenida, integrarla y obtener la posición, en este caso la altura.

$$Velocidad = \frac{\Delta Altura}{\Delta Tiempo} = \frac{Altura - AlturaPrevia}{Tiempo - TiempoPrevio}$$

Con esta fórmula, se obtiene la velocidad del carro como muestra en la figura 6.22.

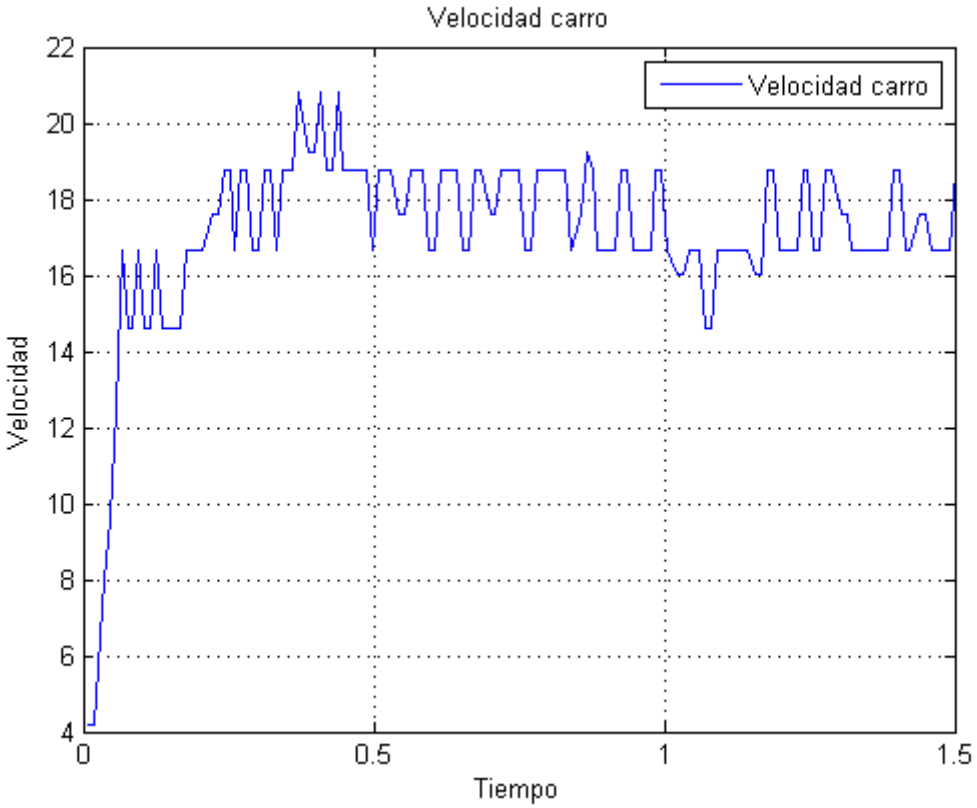


Figura 6.22. Velocidad del carro en lazo abierto.

Como se puede ver en la figura 6.23, la velocidad es similar a un modelo de primer orden.

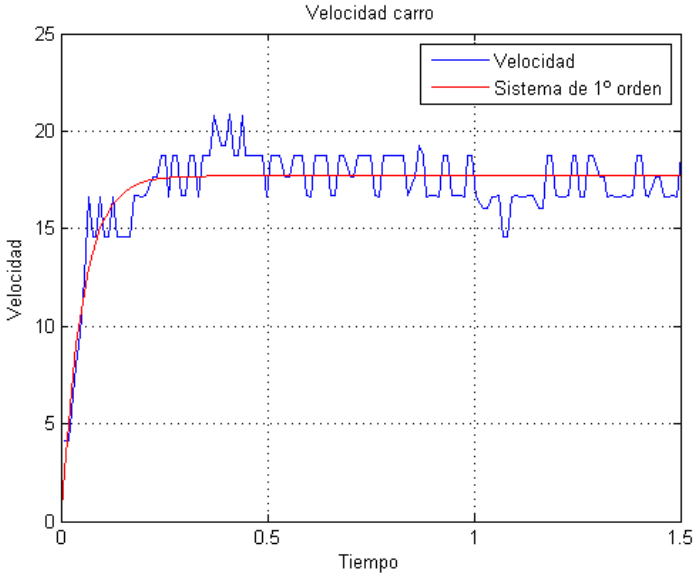


Figura 6.23 Aproximación del modelo a un sistema de primer orden.

El modelo del carro con salida posición se obtiene integrando la velocidad de subida.

Tiempo de establecimiento: $T = 0.05$

Ganancia: $K = 0.18$

Función de transferencia: $G(S) = \frac{3.6}{S+20}$

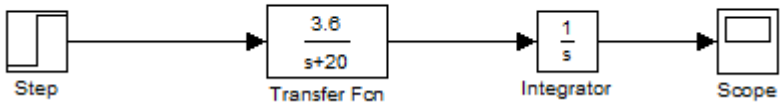


Figura 6.24 Modelo en simulink.

La salida del modelo de posición se muestra en la figura 6.25

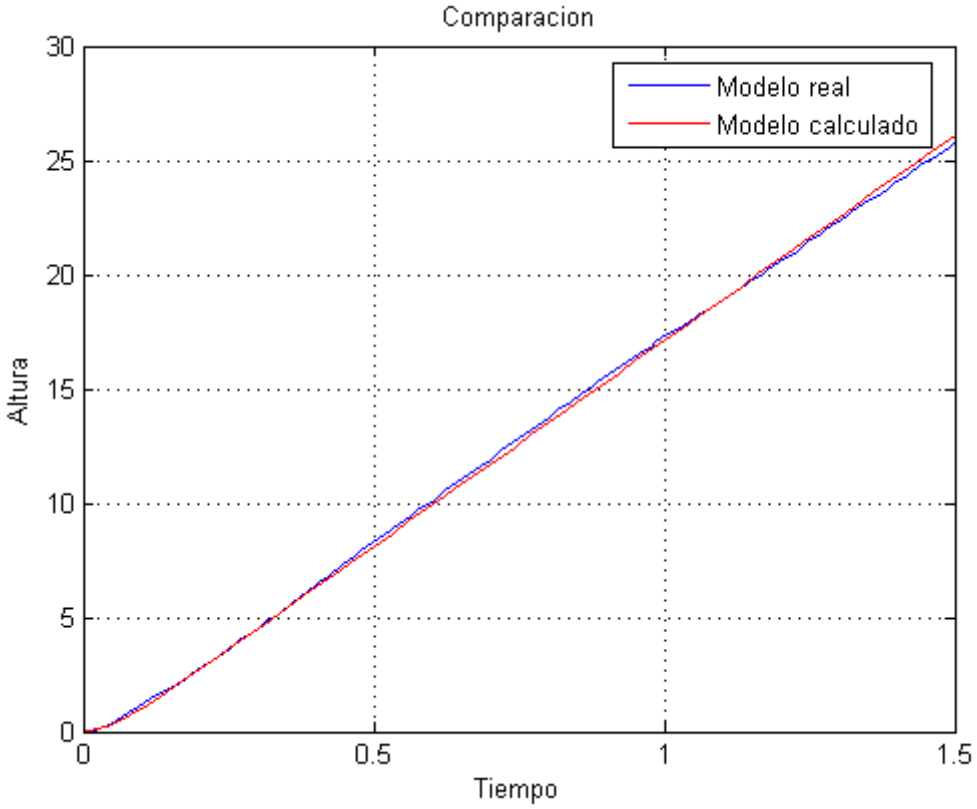


Figura 6.25 Comparación modelo real con modelo calculado.

Si comparamos el modelo calculado con el sistema real, se observa que es un modelo adecuado. El código utilizado para este experimento es incremento.rxe incluido en el Anexo A.

6.4.3 Lazo cerrado.

Se optó por hacer un experimento en lazo cerrado ya que cuando se realizó no se había conseguido todavía la salida mostrada en la figura 6.21.Este experimento contiene un proporcional de valor 6 y se le aplica un escalón de 15. Una vez estabilizado, se le vuelve a aplicar un nuevo escalón, pero esta vez de 10 unidades con el que vamos a trabajar. El código utilizado se puede ver en el Anexo A con el nombre CarroP6.rxe

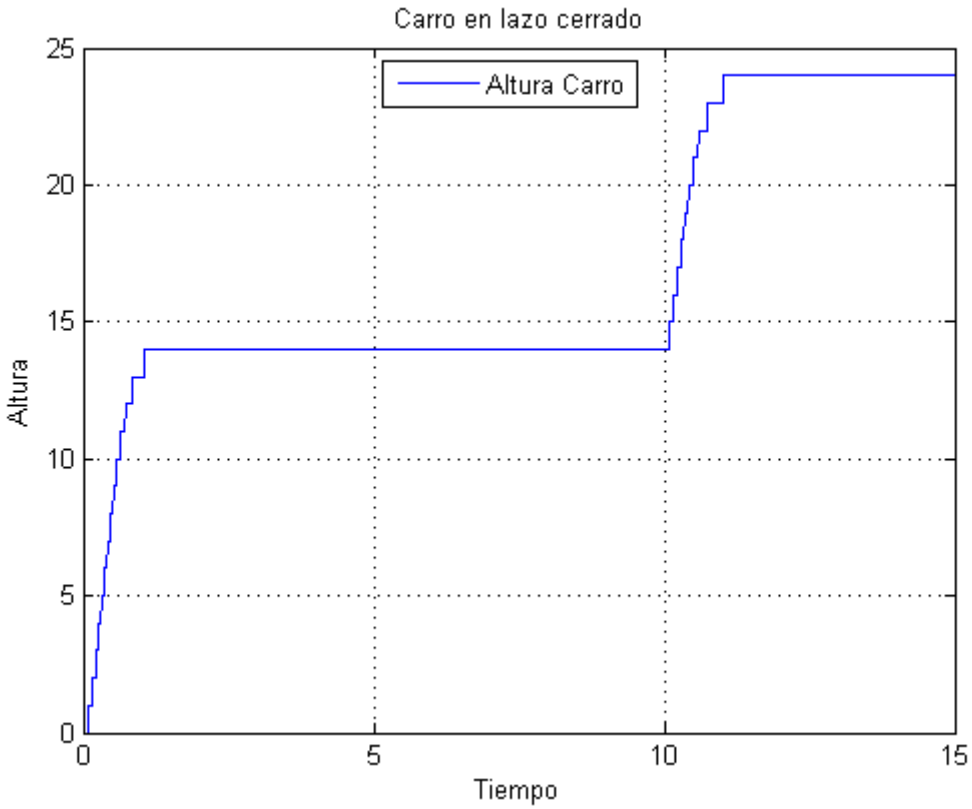


Figura 6.26. Salida lazo cerrado.

Con la salida obtenida mostrada en la figura 6.26, se intentó sacar un modelo de primer orden. En este modelo el tiempo de establecimiento es demasiado corto por lo que para simplificar cálculos, se desprecia T.

$$K = \frac{1}{6}\frac{Y(S)}{U(S)} = \frac{10}{60} = 0.1666$$

Ajustando K manualmente, se obtiene una ganancia de: K=0.175

El modelo tendrá la forma de un integrador: $G(S) = \frac{K}{S} = \frac{0.175}{S}$

Quedando de esta manera:

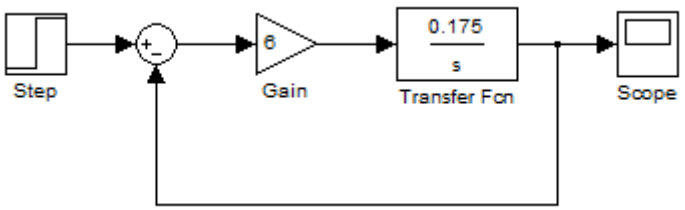


Figura 6.27. Modelo simulink.

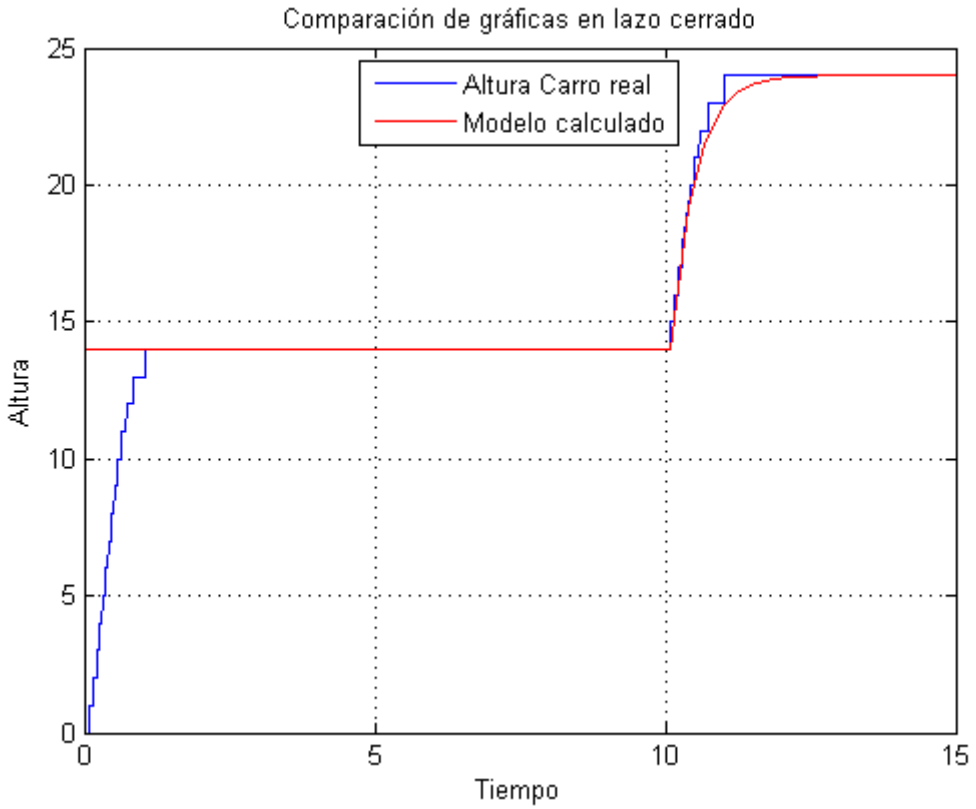


Figura 6.28.Comparación entre modelo real y calculado en lazo cerrado.

Comparando la salida real en lazo cerrado con la salida del modelo calculado, se observa que el modelo es similar y por tanto es adecuado como muestra la figura 6.28.

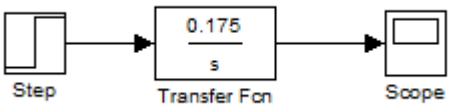


Figura 6.29. Modelo lazo abierto simulink.

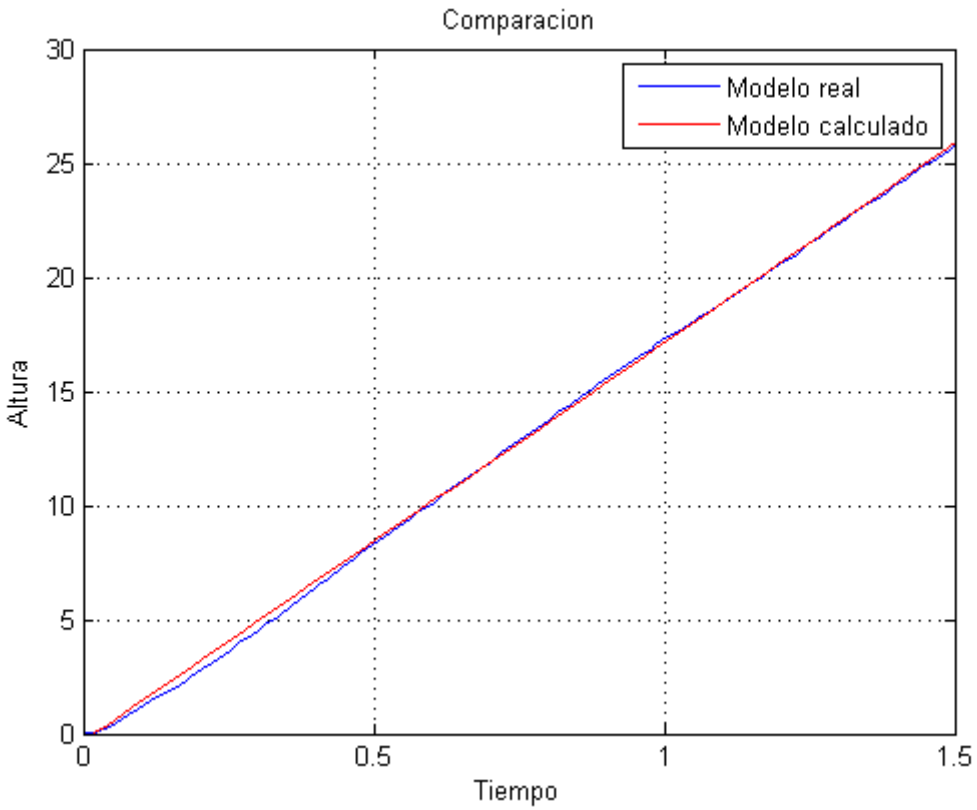


Figura 6.30. Comparación entre modelo real y calculado en lazo abierto.

Comparando el modelo real con el calculado en lazo abierto, se observa que tanto el modelo calculado mediante la velocidad como el modelo calculado en lazo cerrado son similares. Por lo que los dos modelos son adecuados

6.5 Control del carro.

Para el control del carro en simulacion se va a utilizar el primer método de Ziegler-Nichols explicado anteriormente Si al modelo calculado mediante la velocidad se le aplica una entrada en forma de escalón unitaria, se obtiene la figura 6.31

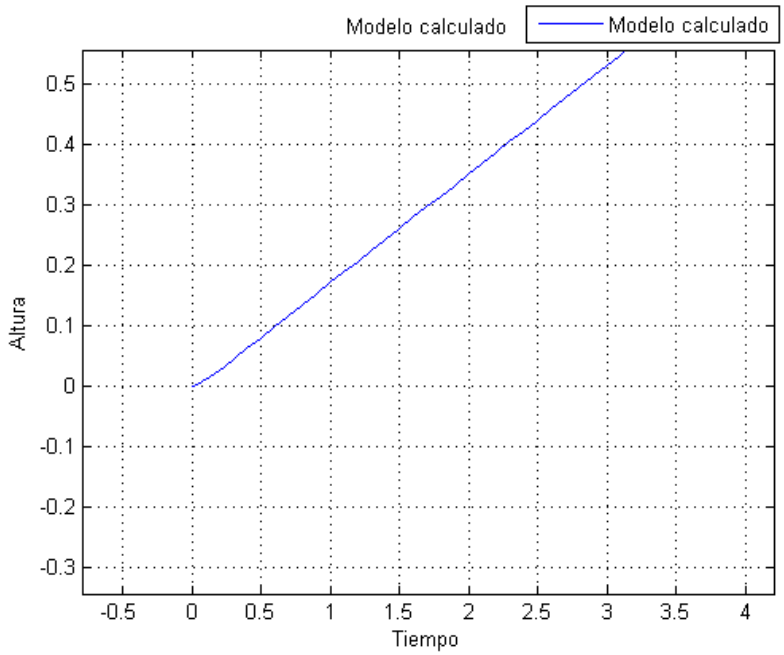


Figura 6.31 Modelo lazo abierto con escalón unitario.

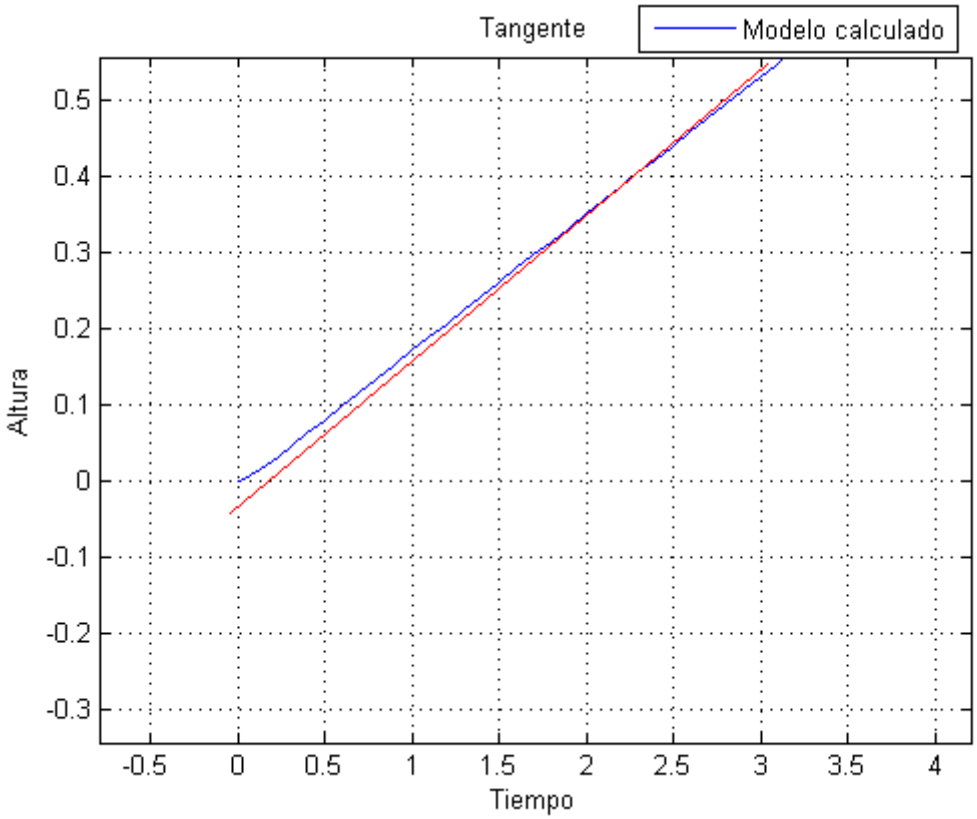


Figura 6.32 Tangente.

Con la salida mostrada en la figura 6.32 se obtiene unos valores de a=0.04 y una L=0.029

Como este sistema contiene un integrador incluido, solo será necesario añadirle un PD. Utilizando la tabla del 1º método de Ziegler-Nichols se obtiene:

$$P = \frac{1.2}{a} = \frac{1.2}{0.04} = 30$$

$$D = 0.5L * K = 0.435$$

Probando estos parametros en simulación:

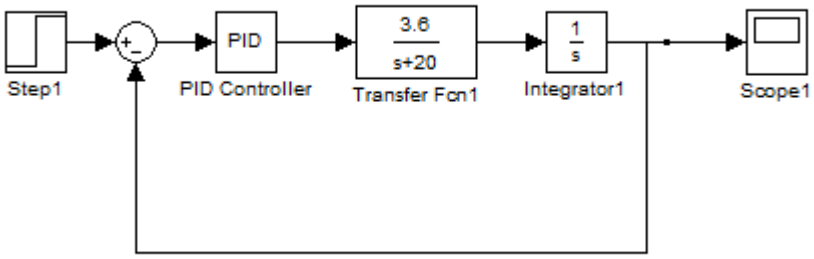


Figura 6.33 Modelo Simulink.

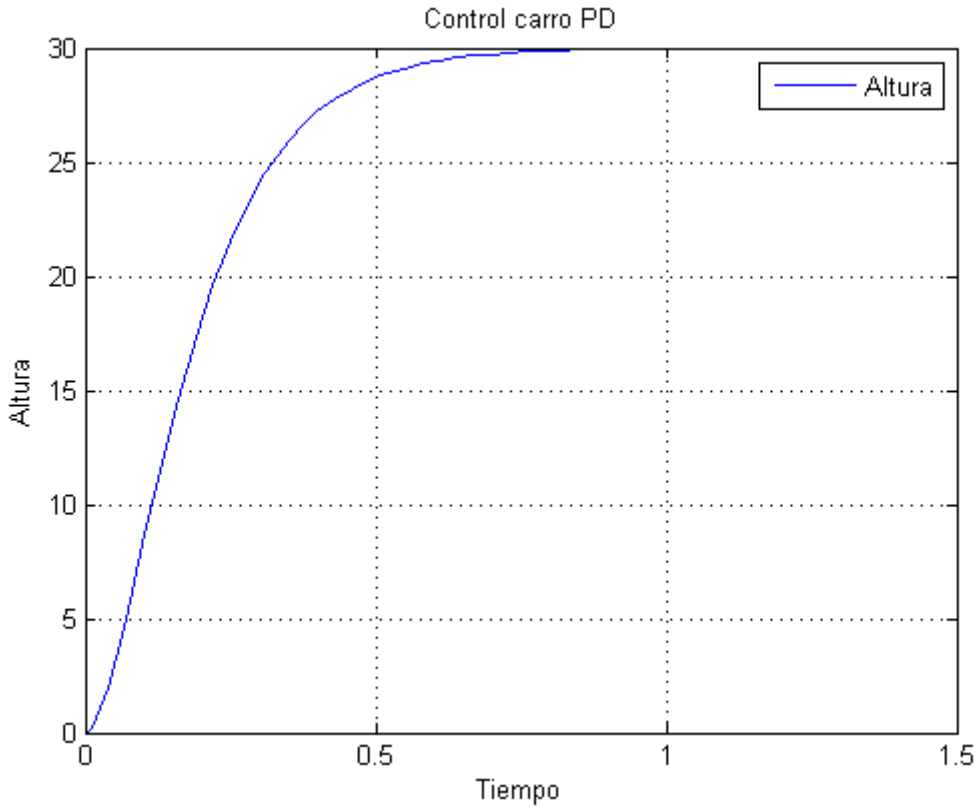


Figura 6.34 Salida simulada controlada con PD.

6.6 Control de la maqueta real.

Cuando se realizó la prueba en la maqueta real, se observó que el carro subía demasiado lento. Se cambió el parámetro D por un valor de D=0.1.Con este nuevo valor se obtuvo la salida mostrada en la figura 6.33.

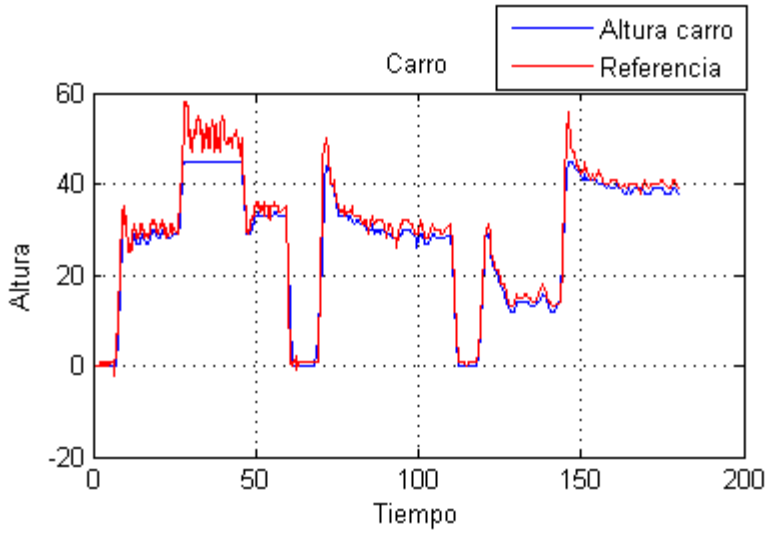


Figura 6.33 Rendimiento del PD

6.7 Control conjunto de la maqueta real.

Para observar el comportamiento que tienen estos controles sobre la maqueta conjuntamente se exponen 4 graficas.

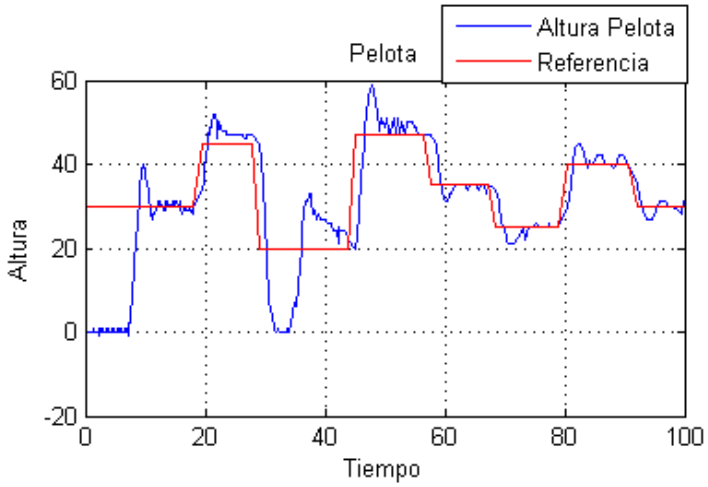


Figura 6.34 Altura pelota

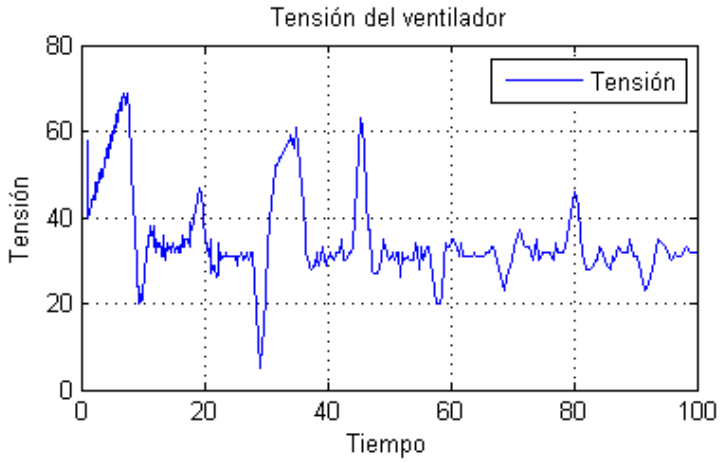


Figura 6.35 Tensión ventilador.

Como se puede observar en la figura 6.34, si se pone una referencia próxima al extremo inferior del tubo, el control no responde de la misma manera ya que este control está realizado para una zona en concreto.

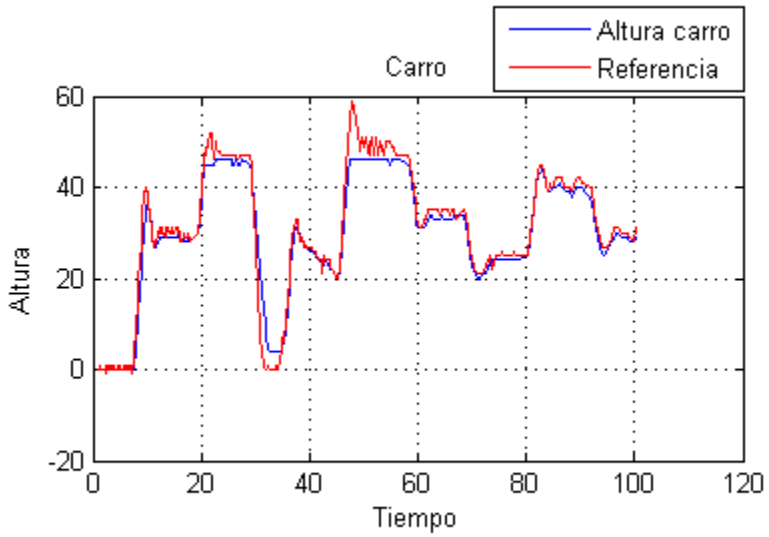


Figura 6.36.Altura carro.

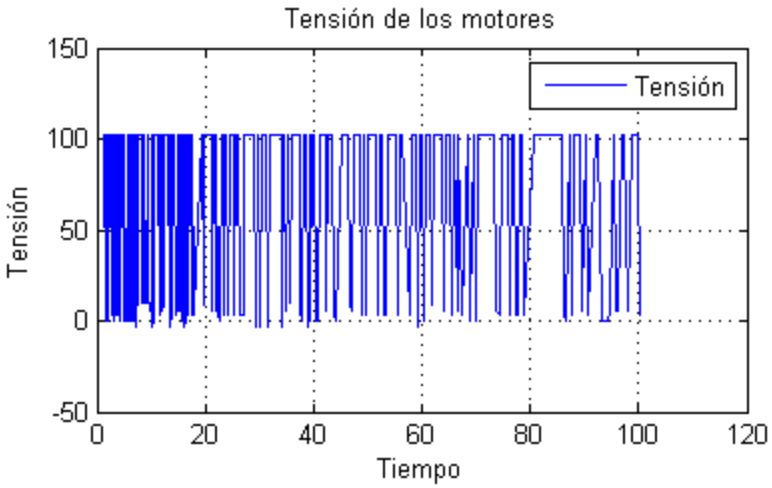


Figura 6.37. Tensión en el motor.

En la figura 6.36 se puede observar la limitación del carro, ya que este no puede superar una altura de 47 cm.

CAPITULO 7: CONCLUSIONES Y TRABAJOS FUTUROS

7.1 Conclusiones

La realización de esta maqueta nos ayuda a poner en práctica y comprender todo lo aprendido teóricamente sobre el control de un sistema.

Podemos destacar diferentes conclusiones:

- Se ha conseguido construir un carro con una estructura físicamente resistente y adaptada a nuestra maqueta
- Se ha analizado todas las comunicaciones entre los diferentes dispositivos de la maqueta incluyendo tiempos y problemas.
- Se ha obtenido el modelo aproximado y el valor aceptable de PID para el control de cada prototipo mediante la realización de varios experimentos.
- Se ha trabajado con los sensores que incluye el NXT y se ha demostrado su eficacia en los sistemas de control.
- Se ha trabajado con dos programas distintos (RobotC y Matlab) programados en diferentes lenguajes y ejecutados a la vez sobre nuestra maqueta, observando su compatibilidad para trabajar simultáneamente.
- Se ha conseguido sincronizar toda la maqueta. Los dos sistemas con NXT mediante conexión bluetooth y estos dos con el PC mediante la conexión USB.
- Se ha explicado y construido una GUI en la cual el usuario puede manejar la maqueta fácilmente sin tener conocimientos de programación.

7.2 Trabajos futuros

- Controlar la pelota mediante la apertura de una válvula colocada en el extremo superior del tubo.
- Mejorar estética del carro y solucionar el problema que tiene el carro al descender.
- Hacer posible enviar valores PID en tiempo real al carro mediante la GUI.
- Construir la maqueta con un solo NXT que haga la función de los dos NXT para utilizar bluetooth sin tener problemas.
- Construir un carro que gire sobre el tubo y que se sitúe en un punto determinado de la pelota.

REFERENCIAS

[1] Ruiz Martínez, Jesús. (2011). *Diseño, construcción y control de una maqueta de lego para el seguimiento de un objetivo móvil*.

[2] Borda Taboada, Junior. (2011). *Diseño de controladores para Lego Mindstorms mediante el toolbox RWTHMindstorms NXT*.

[3] http://www.mindstorms.rwth-aachen.de

[4] http://mindstorms.lego.com/en-us/support/files/Driver.aspx

[5] http://www.mindstorms.rwth-aachen.de

[6] Gasperi, M., Hurbain, P. y Hurbain, I. (2007). *Extreme NXT*.

[7] http://www.techbricks.nl/My-NXT-projects/nxt-cable-connector-tongs.html

[8] http://www.philohome.com/nxtplug/nxtplug.htm

[9] http://www.g7smy.co.uk/?lego/extentions

[10] http://mindstorms.lego.com/en-us/support/files/Driver.aspx

ANEXOS

Anexo A. Programas desarrollados para el funcionamiento de la maqueta en RobotC.

• Contpelota

```
#pragma config(Sensor, S1,    sonar,           sensorSONAR)
#pragma config(Sensor, S3,    sonar3,          sensorSONAR)
#pragma config(Sensor, S4,    sonar2,          sensorSONAR)

#pragma config(Motor,  motorA,          ventilador,    tmotorNormal, openLoop, )


task main(){

//Borramos cualquier archivo existente previamente
ClearTimer(T1);
time1[T1]=0;

int x1=0;
int x2=0;
int x3=0;
int a=0;
int b=0;
int c=0;
int d=0;

float salidatotal=0;


float referencia=0;
float distancia=0;
float error = 0;
float errorprevio = 0;
//Variable para almacenar la suma de todos los errores anteriores
float errorsuma = 0;
//Salida que nos proporcionara cada uno de los terminos del controlador
float salidaP = 0;
float salidaI = 0;
float salidaD = 0;
//(Suma de las 3 salidas anteriores y teniendo en cuenta la ganancia
float y=0;
float s=0;

float P = 1.33;
float I = 0.01;
float D = 0.6;
float k=1;
referencia=30;
```

```
//Creamos el primer archivo
int t=0;



        //Inicializo el timer


////CUERPO DEL PROGRAMA////
//Mientras que no haya objetos la moto sigue avanzando


t=time1[T1];
while(true)//Bucle infinito ya que es la GUI la que detiene el programa.
{

        if ( bQueuedMsgAvailable () )//Recoge los valores del PID enviado desde la
GUI
{
s=messageParm[0];
//nxtDisplayCenteredTextLine(6,"%d",s);
ClearMessage();
if (s==254)//Si el valor recibido es 254,Matlab está a punto de enviar el valor del PID
nuevo.
{
ClearMessage();
until (bQueuedMsgAvailable());
a=messageParm[0];
ClearMessage();
until (bQueuedMsgAvailable());
b=messageParm[0];
ClearMessage();
until (bQueuedMsgAvailable());
c=messageParm[0];
ClearMessage();
until (bQueuedMsgAvailable());
d=messageParm[0];
ClearMessage();

P=a;
I=0.01*b;
D=0.1*c;
referencia=d;
nxtDisplayCenteredTextLine(3,"%f",P);
nxtDisplayCenteredTextLine(4,"%f",I);
nxtDisplayCenteredTextLine(5,"%f",D);
nxtDisplayCenteredTextLine(6,"%f",referencia);
}
}
wait1Msec (50); // wait one -twentieth of a second
```

```
nxtDisplayString(7,"distancia= %f",y);



          distancia=SensorValue[S1];
SensorValue[S4]=distancia;
SensorValue[S3]=salidatotal;


          while (distancia >100)
          {
            //SetSensorType(S1, sensorSONAR);

            distancia=SensorValue[S1];
            SensorValue[S4]=distancia;
            SensorValue[S3]=salidatotal;
          }
         x2=distancia;

          sendMessageWithParm(x1,x2,x3);
          y=66-distancia;


   errorprevio=error;
   error=referencia-y;


                  //PID
   salidaP=error*P;
   errorsuma+=error;
   salidaI=errorsuma*I;
   salidaD=(error-errorprevio)*D;
                  //Salida completa teniendo en cuenta la ganancia K


   salidatotal=k*(salidaP+salidaI+salidaD);
//Saturacion angulo maximo que puede girar
       if(salidatotal>75)
   {
     salidatotal=75;
     }
   else if(salidatotal<0)
   {
         salidatotal=0;
    }
    else
       {
salidatotal=salidatotal;
    }
```

```
motor[ventilador]=salidatotal;
SensorValue[S3]=salidatotal;//Asignamos el valor de salida total al puerto del sensor 3


wait1Msec(10);
}

}
```

• CarroPID.

```
#pragma config(Sensor, S2,     luz,               sensorLightActive)

#pragma config(Motor,  motorA,          uno,           tmotorNormal, openLoop, )
#pragma config(Motor,  motorB,          dos,           tmotorNormal, openLoop, )

task main()
{
        int a=0;
        int b=0;
        int c=0;
        int s=0;
        int d=0;
        int rt=0;
int t=0;
int pasarela=0;
int q=0;
int w=0;
int posicarro=0;
float P=20;
float I=0;
float D=0;
float error=0;
float errorprevio=0;
float errorsuma=0;
float SalidaP;
float SalidaI;
float SalidaD;
float SalidaTotal=0;
float k=1;
int referencia=0;
int control=0;
int luzmin=0;

//Reproducimos un sonido de aviso
PlayTone(400, 20);
```

```
luzmin=(SensorValue[luz]-1);//Coge el valor de luz que hay en el momento iniciar para saber cuándo tiene que resetear el encoder más adelante.
ClearTimer(T1);
time1[T1]=0;

while (true)//Bucle infinito ya desde la GUI se detendrá este programa
{
t=time1[T1];



if (bQueuedMsgAvailable())
{
s=messageParm[0];
rt=messageParm[1];
nxtDisplayCenteredTextLine(6,"%d",s);
if (s==254)//Si el mensaje recibido es 254 quiere decir que Matlab va a enviar valores del PID pero esta parte no se utiliza ya que si el NXT envía el valor de la altura mientras se recibe el PID se mezclaran y obtendremos un valor del PID no deseado. Por lo que siempre que reciba un valor siempre debe ser del NXTA. Explicado en el Anexo C1.
{
ClearMessage();
until (bQueuedMsgAvailable());
a=messageParm[0];
ClearMessage();
until (bQueuedMsgAvailable());
b=messageParm[0];
ClearMessage();
until (bQueuedMsgAvailable());
c=messageParm[0];
ClearMessage();
until (bQueuedMsgAvailable());

P=a;
I=0.01*b;
D=0.1*c;

nxtDisplayCenteredTextLine(3,"%f",P);
nxtDisplayCenteredTextLine(4,"%f",I);
nxtDisplayCenteredTextLine(5,"%f",D);
}
else
{

referencia=66-rt;

ClearMessage();
}}
if (referencia>47)//Limitamos el carro a una altura, ya que si sobrepasa ese valor se saldrá del tubo.
```

```
{

referencia=47;
}
        q=-nMotorEncoder[dos];
        w=q/24;//Pasamos el valor del encoder a centimetros
        posicarro=w;
SensorValue[S4]=w;//Asignamos al puerto 4 el valor de w

if (referencia<2)//Si la pelota está mas debajo de dos centímetros, y el valor del sensor de luz es mayor que luzmin, reseteará el encoder.
{

if (SensorValue[luz]>luzmin)
{

nMotorEncoder[dos]=0;//Con esto corregimos el error que se haya producido .
}
}


//PID
      errorprevio=error;
      error=referencia-w;
      SalidaP=error*P;
      errorsuma+=error;
      SalidaI=errorsuma*I;
      SalidaD=(error-errorprevio)*D;
      SalidaTotal=k*(SalidaP+SalidaI+SalidaD);
//Muestra por pantalla el PID
      nxtDisplayCenteredTextLine(3,"%f",P);
nxtDisplayCenteredTextLine(4,"%f",I);
nxtDisplayCenteredTextLine(5,"%f",D);
//Limita el motor del carro ya que si sube no puede sobrepasar el valor 100 y si baja no debe de ir más rápido de 35 ya que si no bajaría demasiado rápido y podría romperse el carro .Hacia arriba debe vencer su peso y hacia abajo su peso proporciona más fuerza.
   if (SalidaTotal>100)
      {
         SalidaTotal=100;
      }
      if (SalidaTotal<0)
      {
         SalidaTotal=-35;
      }
      nxtDisplayCenteredTextLine(6,"%f",SalidaTotal);
nxtDisplayCenteredTextLine(7,"%d",referencia);
SensorValue[S3]=SalidaTotal;//Asigna el valor salidatotal al puerto 3 del NXT
      motor[uno]=-SalidaTotal;          // Asigna el valor salidatotal a los motores.
          motor[dos]=-SalidaTotal;
```

```
      wait1Msec(1);
    }



}
```

• CBluetooth.

```
task main()
{
        setBluetoothOn();//Pone el bluetooth en ON
        wait10Msec(250);
        btConnect(2, "NXTB");//Conecta con NXTB
        wait10Msec(20);


}
```

• Apagar.

```
task main()
{

powerOff();//Apaga el NXT

}
```

• Tensión_Motor.

```
#pragma config(Motor,  motorA,          uno,          tmotorNormal, openLoop, )
#pragma config(Motor,  motorB,          dos,          tmotorNormal, openLoop, )

TFileIOResult resultado; //Flag que nos indica el resultado de la operacion con un
archivo
TFileHandle etiqueta; //Etiqueta del archivo
int longitud = 25000; //Longitud de cada archivo en bytes
string nombrearchivo; //String que sera el nombre de cada archivo
float altprevia=0;
float a=0;
float b=0;
float c=0;
float d=0;
task main ()
{


        nombrearchivo="altura.txt";
  string datosalida;
  Delete(nombrearchivo, resultado);
  OpenWrite(etiqueta, resultado,nombrearchivo, longitud);
```

```
        nMotorEncoder[uno]=0;
        nMotorEncoder[dos]=0;

        ClearTimer(T1);
 time1[T1]=0;

    motor[uno]=-100;
        motor[dos]=-100;
   while (time1[T1]<=2000)
   {

        datosalida=-nMotorEncoder[dos]/24;
        WriteText(etiqueta,resultado,datosalida+"\t");
        datosalida=-nMotorEncoder[uno]/24;
        WriteText(etiqueta,resultado,datosalida+"\t");
        datosalida=time1[T1];
        WriteText(etiqueta,resultado,datosalida+"\t");
    altprevia=a;
        a=-nMotorEncoder[dos]/24;
        b=a-altprevia;
        datosalida=b;
        WriteText(etiqueta,resultado,datosalida+"\t");
    c=d;
    d=time1[T1];
        datosalida=d-c;
        WriteText(etiqueta,resultado,datosalida+"\r\n");
   // wait1Msec(10);


   }
CloseAllHandles(resultado);
}
```

• CarroP6.

```
#pragma config(Sensor, S2,     luz,              sensorLightActive)

#pragma config(Motor,  motorA,          uno,          tmotorNormal, openLoop, )
#pragma config(Motor,  motorB,          dos,          tmotorNormal, openLoop, )

TFileIOResult resultado; //Flag que nos indica el resultado de la operacion con un
archivo
TFileHandle etiqueta; //Etiqueta del archivo
int longitud = 25000; //Longitud de cada archivo en bytes
string nombrearchivo; //String que sera el nombre de cada archivo
```

```
task main()
{
	int a=0;
	int b=0;
	int c=0;
	int s=0;
	int d=0;
	int rt=0;
int t=0;
int pasarela=0;
int q=0;
int w=0;
int posicarro=0;
float P=6;
float I=0;
float D=0;
float error=0;
float errorprevio=0;
float errorsuma=0;
float SalidaP;
float SalidaI;
float SalidaD;
float SalidaTotal=0;
float k=1;
int referencia=15;
int control=0;

nombrearchivo="asencillo.txt";
string datosalida;
Delete(nombrearchivo, resultado);

OpenWrite(etiqueta, resultado,nombrearchivo, longitud);
//Reproducimos un sonido de aviso
PlayTone(400, 20);

ClearTimer(T1);
time1[T1]=0;

while (time1[T1]<=15000)
{
t=time1[T1];

if (t>10000)
{
	referencia=25;
}

	q=-nMotorEncoder[dos];
	w=q/24;
	posicarro=w;
```

```
  pasarela=posicarro;                          //el archivo de texto
  datosalida=pasarela;
  WriteText(etiqueta,resultado,datosalida+"\t");

  pasarela=SalidaTotal;                        //el archivo de texto
  datosalida=pasarela;
  WriteText(etiqueta,resultado,datosalida+"\t");

  pasarela=t;                          //el archivo de texto
  datosalida=pasarela;
  WriteText(etiqueta,resultado,datosalida+"\t");

  pasarela=referencia;
        datosalida=pasarela;
  WriteText(etiqueta,resultado,datosalida+"\r\n");
  wait10Msec(1);



      errorprevio=error;
      error=referencia-w;//posiblemente haya aqui un fallo
      SalidaP=error*P;
      errorsuma+=error;
      SalidaI=errorsuma*I;
      SalidaD=(error-errorprevio)*D;
      SalidaTotal=k*(SalidaP+SalidaI+SalidaD);

      nxtDisplayCenteredTextLine(3,"%f",P);
nxtDisplayCenteredTextLine(4,"%f",I);
nxtDisplayCenteredTextLine(5,"%f",D);
      Salidatotal+=50;
   if (SalidaTotal>100)
      {
         SalidaTotal=100;
      }
      if (SalidaTotal<0)
      {
         SalidaTotal=-35;
      }
      nxtDisplayCenteredTextLine(6,"%f",SalidaTotal);
nxtDisplayCenteredTextLine(7,"%d",referencia);
      motor[uno]=-SalidaTotal;          // carro arriba
          motor[dos]=-SalidaTotal;
      wait1Msec(1);
     }
```

```
}
```

• Estructura PID

```
float error = 0;
float errorprevio = 0;
float errorsuma = 0;
float salidaP = 0;
float salidaI = 0;
float salidaD = 0;
float k=1;
float P=0;
float I=0;
float D=0;
float k=0;
salidaP=error*P;
errorsuma+=error;
salidaI=errorsuma*I;
salidaD=(error-errorprevio)*D;
//Salida completa teniendo en cuenta la ganancia K

   salidatotal=k*(salidaP+salidaI+salidaD);
```

• Escribir en un documento TXT

```
TFileIOResult resultado; //Flag que nos indica el resultado de la operacion con un archivo
TFileHandle etiqueta; //Etiqueta del archivo
int longitud = 25000; //Longitud de cada archivo en bytes
string nombrearchivo; //String que sera el nombre de cada archivo

task main ()
{


        nombrearchivo="encoder.txt";
 string datosalida;
 Delete(nombrearchivo, resultado);
 OpenWrite(etiqueta, resultado,nombrearchivo, longitud);




ClearTimer(T1);
 time1[T1]=0;

   while (time1[T1]<=10000)
  {
```

```
        datosalida=-nMotorEncoder[dos];
        WriteText(etiqueta,resultado,datosalida+"\t");
        datosalida=time1[T1];
        WriteText(etiqueta,resultado,datosalida+"\r\n");
        wait10Msec(1);
   }
CloseAllHandles(resultado);
}
```

• Incremento

```
#pragma config(Motor,  motorA,          uno,           tmotorNormal, openLoop, )
#pragma config(Motor,  motorB,          dos,           tmotorNormal, openLoop, )

TFileIOResult resultado; //Flag que nos indica el resultado de la operacion con un
archivo
TFileHandle etiqueta; //Etiqueta del archivo
int longitud = 25000; //Longitud de cada archivo en bytes
string nombrearchivo; //String que sera el nombre de cada archivo

task main ()
{

int a=0;
int b=0;
int c=0;
int d=0;
        nombrearchivo="inc.txt";
 string datosalida;
 Delete(nombrearchivo, resultado);
 OpenWrite(etiqueta, resultado,nombrearchivo, longitud);


        ClearTimer(T1);
 time1[T1]=0;

   while (time1[T1]<=1500)
   {
   b=a;
   a=-nMotorEncoder[dos];
   d=c;
   c=time1[T1];
        motor[uno]=-100;
        motor[dos]=-100;
        datosalida=a;
        WriteText(etiqueta,resultado,datosalida+"\t");
        datosalida=b;
        WriteText(etiqueta,resultado,datosalida+"\t");
        datosalida=c;
        WriteText(etiqueta,resultado,datosalida+"\t");
```

```
        datosalida=d;
        WriteText(etiqueta,resultado,datosalida+"\r\n");
        wait10Msec(1);


    }
CloseAllHandles(resultado);
}
```

Anexo B Programas desarrollados para el funcionamiento de la maqueta Matlab.

•Realizar Conexión: Muestra un mensaje para comenzar a realizar la conexión.
Si contestamos que sí, se nos habilitaran los botones de conexión del NXTA y NXTB

```
function pushbutton6_Callback(hObject, eventdata, handles)

opc=questdlg('¿Desea comenzar con el asistente para la conexión de los
NXT?','CONEXIÓN','Si','No','No')

if strcmp(opc,'No')
    return;
end
COM_CloseNXT all;

msgbox('Conecte NXT_Pelota y NXT_Carro a un puerto USB',' CONEXIÓN ')
set(handles.pushbutton2,'Enable','on');
set(handles.pushbutton5,'Enable','on');
```

•Conectar NXTA: Conecta el Ladrillo NXTA al PC para poder utilizarlo en la GUI.

```
function pushbutton2_Callback(hObject, eventdata, handles)
x=10;
while (x) %Bucle infinito
hNXTA=COM_OpenNXTEX('USB','0016530EBEB9');
pause(1);
handles.hNXTA=hNXTA;
guidata(hObject,handles)
NXT_PlayTone(200,500,hNXTA);
pause(0.3);
opc1=questdlg('¿Ha      escuchado      el      sonido      en      el
NXT_Pelota?','COMPROBACIÓN','Si','No','No')
   if strcmp(opc1,'Si')
      msgbox('Conexion    NXT_Pelota    establecida    correctamente','
CONEXIÓN NXT_Pelota ')

      set(handles.pushbutton13,'Enable','on');  %Habilita  el  boton  de
conectar por bluetooth.

      break;
   end
end
```

•Conectar NXTB: Conecta el Ladrillo NXTB al PC para poder utilizarlo en la GUI.

```
function pushbutton5_Callback(hObject, eventdata, handles)
x=10;

while (x) %Bucle infinito

  hNXTB=COM_OpenNXTEX('USB','0016530EC2B3'); %Conecta mediante su MAC
   pause(1);
handles.hNXTB=hNXTB;
guidata(hObject,handles)
NXT_PlayTone(200,500,hNXTB);
```

```
   pause(0.3);
   opc2=questdlg('¿Ha escuchado el sonido en el
NXT_Carro?','COMPROBACIÓN','Si','No','No')
   if strcmp(opc2,'Si')
    msgbox('Conexion NXT_Carro establecida correctamente',' CONEXIÓN
NXT_Carro ')

    break;
   end
end
```

•Conectar por Bluetooth: Conecta el NXTA al NXTB por bluetooth ejecutando el archivo cbluetooth.rxe en el NXTA.

```
while (x)
hNXTA=handles.hNXTA;
NXT_StopProgram(hNXTA);
 pause(1.5)
NXT_StartProgram('cbluetooth.rxe', hNXTA);
pause(7);
   opc2=questdlg('¿Ha cambia el simbolo < a este otro <> en la esquina
superior izquierda de los dos NXT ?','COMPROBACIÓN','Si','No','No')
   if strcmp(opc2,'Si')
    opc=msgbox('Conexion establecida correctamente',' CONEXION ')

    set(handles.pushbutton30,'Enable','on');   %Habilita   el   boton
Establecer tiempo

    set(handles.pushbutton8,'Enable','on'); %Habilita el boton Apagar
y salir.


    break;
   end
end
```

•Establecer tiempo: Con este código elegimos el tiempo que queremos que dure nuestra prueba.

```
function pushbutton30_Callback(hObject, eventdata, handles)

n=0
b=0

valor=str2double(get(handles.edit13,'String'));

if (valor<0)
    errordlg('El valor debe ser positivo','ERROR'); % Comprueba que
sea positivo
    n=1;
end
if isnan(valor)
errordlg('El valor debe ser numérico','ERROR'); %Comprueba que será
numérico
n=1;
end
if (n==0)
```

```
b=str2double(get(handles.edit13,'String')); %Coge el tiempo del edit13
set(handles.text21,'String',b); %Manda el texto a un text21 de donde
lo cogera el botón comenzar
set(handles.pushbutton7,'Enable','on'); %Habilita el botón comenzar
end
```

•Comenzar: Ejecuta los programas CarroPID.rxe en el NXTB y ContPelota.rxe en el NXTA. Muestra las gráficas en tiempo real de la altura de la bola, tensión del ventilador, altura del carro y tensión del motor del carro.

```
% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
set(handles.pushbutton15,'Enable','off');   %Deshabilita   el   botón
guardar valores
set(handles.pushbutton8,'Enable','off'); %Deshabilita el botón apagar
y salir
set(handles.pushbutton1,'Enable','on');  %Habilita  el  botón  enviar
valores

axes(handles.axes1); %Selecciona un Axes y elimina su contenido.
cla reset;
axes(handles.axes2);
cla reset;
axes(handles.axes3);
cla reset;
axes(handles.axes4);
cla reset;

hNXTB=handles.hNXTB;
NXT_StopProgram(hNXTB); %Si en el NXTB se esta ejecutando un programa,
este comando lo detiene.
 pause(1.5)

 OpenUltrasonic(SENSOR_4,'SINGLE_SHOT',hNXTB); %Abre el sensor 4 donde
recibe los valores del encoder.
 NXT_StartProgram('carroPID.rxe', hNXTB); %Ejecuta el programa en el
NXTB

hNXTA=handles.hNXTA;
 NXT_StopProgram(hNXTA);
 pause(1.5)
OpenUltrasonic(SENSOR_3,'SINGLE_SHOT',hNXTA); %Abre el sensor 3 del
NXTA donde recibe los valores de tension del ventilador.
 OpenUltrasonic(SENSOR_4,'SINGLE_SHOT',hNXTA); %Abre el sensor 4 del
NXTA para recibir los valores de la altura de la bola.
 pause(1);
NXT_StartProgram('contpelota.rxe', hNXTA); %Ejecuta el programa en el
NXTA.



salir=0;
handles.salir=salir;
%Declaramos las variables a utilizar.
temp=str2double(get(handles.text21,'String')); %Obtiene el tiempo del
text21
```

```
a=0;
b=0;
c=0;
d=0;
e=0;
g=0;
h=0;
y=0;
x=0;
n=0;
f=0;
i=0;
r=0;
t=0;
z=0;
m=0;
l=0;
u=0;
data=0;
data1=0;
data2=0;



tic; %Inicia una variable que mide el tiempo
while (toc<temp) %Este bucle durará el valor de temp en segundos
n=n+1;

%/////////////////Sensor de ultrasonidos////////////////grafica 1
a=toc; %Le asignamos el tiempo

data=NXT_GetInputValues(SENSOR_4, hNXTA); %Leemos el sensor 4 del NXTA
b=data.ScaledVal; %Leemos la parte que nos interesa.
while (b>70)
%Este bucle se utiliza por si cuando leamos nos falsea la medida,que
vuelva a repetir el paso anterior antes de poner el dato obtenido en
la grafica.
    OpenUltrasonic(SENSOR_4,'SINGLE_SHOT',hNXTA);
  data=NXT_GetInputValues(SENSOR_4, hNXTA);
b=data.ScaledVal; %Obtiene la altura de la pelota.
end

f=66-b; %Se le resta 66 para tener la altura de la bola en centímetros
x(n)=a;
y(n)=f;
z(n)=str2double(get(handles.textinv,'String'));
axes(handles.axes1); %Seleccionamos el axes1
handles.h=plot(x,y); %Se le asigna el plot.

set(handles.h,'Color','b');grid on; %Añade el plot al Axes1
if (n==1)
title('Pelota') %Título
xlabel('Tiempo') % Etiqueta el eje horizontal
ylabel('Altura') % Etiqueta el eje vertical
end
hold on
handles.h=plot(x,z);
set(handles.h,'Color','r');grid on;
```

```
%/////////////////////tension salida ventilador/////////////grafica 2
a=toc;
data2=NXT_GetInputValues(SENSOR_3, hNXTA);
i=data2.ScaledVal;
while (i>70)
  data2=NXT_GetInputValues(SENSOR_3, hNXTA);
i=data2.ScaledVal;
end

r(n)=a;
t(n)=i;
axes(handles.axes2);
handles.h2=plot(r,t);
set(handles.h2,'Color','b');grid on;
title('Tension Salida Ventilador') %Título
xlabel('Tiempo') % Etiqueta el eje horizontal
ylabel('Tension') % Etiqueta el eje vertical




%/////////////////Encoder/////////////////grafica 3
axes(handles.axes3);
a=toc;
data3=NXT_GetInputValues(SENSOR_4, hNXTB);
d=data3.ScaledVal;
while (d>70)
    OpenUltrasonic(SENSOR_4,'SINGLE_SHOT',hNXTB);
  data3=NXT_GetInputValues(SENSOR_4, hNXTB);
d=data3.ScaledVal;
end
g(n)=a;
h(n)=d;

handles.h3=plot(g,h);
set(handles.h3,'Color','b');grid on;
title('Carro') %Título
xlabel('Tiempo') % Etiqueta el eje horizontal
ylabel('Altura') % Etiqueta el eje vertical
hold on
handles.h3=plot(g,y);
set(handles.h3,'Color','r');grid on;

%/////////////////Tension salida motor carro/////grafica 4

axes(handles.axes4);
a=toc;
data4=NXT_GetInputValues(SENSOR_3, hNXTB);
l=data4.ScaledVal;
e(n)=a;
u(n)=l;

handles.h3=plot(e,u);
set(handles.h3,'Color','b');grid on;

title('Tension Salida Motores') %Título
xlabel('Tiempo') % Etiqueta el eje horizontal
ylabel('Tension') % Etiqueta el eje vertical
```

```
salir=handles.salir;
if (salir==1)
    CloseSensor(SENSOR_1, hNXTA);
    CloseSensor(SENSOR_2, hNXTB)
   break;
end
end
axes(handles.axes1);
legend('Altura bola', 'Referencia') % Pone una leyenda
axes(handles.axes2);
legend('VsalidaVentilador')
set(handles.pushbutton15,'Enable','on');
axes(handles.axes3);
legend('Altura carro', 'Altura bola')
axes(handles.axes4);
legend('VsalidaMotor')
set(handles.pushbutton1,'Enable','off');  %Habilita  y  deshabilita  al
contrario que al inicio
set(handles.pushbutton15,'Enable','on');
set(handles.pushbutton8,'Enable','on');
hNXTB=handles.hNXTB;
NXT_StopProgram(hNXTB); %Detiene el programa de RobotC en el NXTB
pause(1.5)
hNXTA=handles.hNXTA;
NXT_StopProgram(hNXTA); %Detiene el programa de RobotC en el NXTA
pause(1.5)
```

•Apagar y salir: Apaga NXTA y NXTB mediante la ejecución del programa apagar.rxe.
Cierra la interfaz.

```
opc=questdlg('¿Desea salir del programa?','SALIR','Si','No','No');
if strcmp(opc,'No')
return;
end
hNXTB=handles.hNXTB;
 NXT_StartProgram('apagar.rxe', hNXTB); %Ejecuta apagar.rxe
 pause(1.5)
 hNXTA=handles.hNXTA;
 NXT_StartProgram('apagar.rxe', hNXTA);
 pause(1.5)
```

•Menu: En el elegimos a quien queremos enviar los valores en tiempo real del PID.

```
  switch get(handles.popupmenu1,'Value')

      case 2
          set(handles.edit1,'Enable','on');
          set(handles.edit2,'Enable','on');
          set(handles.edit3,'Enable','on');
          set(handles.edit4,'Enable','on');
%se habilitan los cuadros de texto para el caso del NXT_Pelota
```

```
      case 3
          set(handles.edit4,'String','0');
          set(handles.edit1,'Enable','off');
          set(handles.edit2,'Enable','off');
          set(handles.edit3,'Enable','off');
        set(handles.edit4,'Enable','off');
        set(handles.edit1,'String','0');
        set(handles.edit2,'String','0');
        set(handles.edit3,'String','0');
  %Se deshabilitan los cuadros de texto ya que no se pueden utilizar
debido a un inconveniente explicado en Anexo C1.

    otherwise
end
```

•Enviar valores: Envía los valores de PID y referencia (en el caso del NXTA) una vez seleccionado a quien enviar en el menú y que valores de PID y referencia deseamos.

```
function pushbutton1_Callback(hObject, eventdata, handles)

valorcomprobacion=0
valorp=  str2double(get(handles.edit1,'String'));  %Obtiene  el  texto
escrito en el elemento edit1.
valori= str2double(get(handles.edit2,'String'));
valord= str2double(get(handles.edit3,'String'));
referencia= str2double(get(handles.edit4,'String'));
handles.referencia=referencia;
if  (valorp<0)||(valori<0)||(valord<0)||(referencia<0)  %Comprueba  que
los valores sean positivos.
    valorcomprobacion=1
errordlg('Los valores deben ser positivos','ERROR') %Mensaje de error.
end
if       isnan(valorp)||isnan(valori)||isnan(valord)||isnan(referencia)
%Comprueba que el texto introducido sea numérico.
 valorcomprobacion=1
 errordlg('Los valores deben ser numéricos','ERROR')
end
if valorcomprobacion==0

  switch get(handles.popupmenu1,'Value')

    case 2%Caso para NXTA(Pelota)
        hNXTA=handles.hNXTA;
        NXT_MessageWrite(254,0,hNXTA)
        pause(0.2);
        NXT_MessageWrite(valorp,0,hNXTA);
        pause(0.2);
        NXT_MessageWrite(valori,0,hNXTA);
        pause(0.2);
        NXT_MessageWrite(valord,0,hNXTA);
        pause(0.2);
        NXT_MessageWrite(referencia,0,hNXTA);
        pause(0.2);
        set(handles.textinv,'String',referencia);
    case 3%Caso para NXTB(Carro)
  %No haría nada ya que no se va a utilizar. Explicado en el Anexo C1.
    otherwise
end
```

```
end
```

•Guardar graficas: Este botón sirve para guardar las gráficas obtenidas en el programa en formato figura.

```
function pushbutton15_Callback(hObject, eventdata, handles)
cd c:\ %Seleccionamos el directorio C:
mkdir('PFC');  %Crea  una  carpeta  llamada  PFC  en  el  directorio
seleccionado
cd c:\PFC %Seleccionamos esa carpeta

%////////////Grafica1///////////////
figura = figure;
% Unidades y posición
unidades  = get(handles.axes1,'Units');
posicion   = get(handles.axes1,'Position');
objeto_2  = copyobj(handles.axes1,figura);
% Modificar la nueva figura
set(objeto_2,'Units',unidades);
set(objeto_2,'Position',[15 5 posicion(3) posicion(4)]);
% Ajustar la nueva figura
set(figura,'Units',unidades);
set(figura,'Position',[15 5 posicion(3)+30 posicion(4)+10]);
% Guardar la gráfica
saveas(figura,'Grafica1_Altura_Pelota')
%Cerrar figura
close(figura)

%////////////Grafica2///////////////
figura = figure;
% Unidades y posición
unidades  = get(handles.axes2,'Units');
posicion   = get(handles.axes2,'Position');
objeto_2  = copyobj(handles.axes2,figura);
% Modificar la nueva figura
set(objeto_2,'Units',unidades);
set(objeto_2,'Position',[15 5 posicion(3) posicion(4)]);
% Ajustar la nueva figura
set(figura,'Units',unidades);
set(figura,'Position',[15 5 posicion(3)+30 posicion(4)+10]);
% Guardar la gráfica
saveas(figura,'Grafica2_Tension_Ventilador')
%Cerrar figura
close(figura)

%////////////Grafica3///////////////
figura = figure;
% Unidades y posición
unidades  = get(handles.axes3,'Units');
posicion   = get(handles.axes3,'Position');
objeto_2  = copyobj(handles.axes3,figura);
% Modificar la nueva figura
set(objeto_2,'Units',unidades);
set(objeto_2,'Position',[15 5 posicion(3) posicion(4)]);
% Ajustar la nueva figura
set(figura,'Units',unidades);
set(figura,'Position',[15 5 posicion(3)+30 posicion(4)+10]);
% Guardar la gráfica
saveas(figura,'Grafica3_Encoder_Carro')
%Cerrar figura
```

```
close(figura)

%////////////Grafica4///////////////
figura = figure;
% Unidades y posición
unidades  = get(handles.axes4,'Units');
posicion   = get(handles.axes4,'Position');
objeto_2  = copyobj(handles.axes4,figura);
% Modificar la nueva figura
set(objeto_2,'Units',unidades);
set(objeto_2,'Position',[15 5 posicion(3) posicion(4)]);
% Ajustar la nueva figura
set(figura,'Units',unidades);
set(figura,'Position',[15 5 posicion(3)+30 posicion(4)+10]);
% Guardar la gráfica
saveas(figura,'Grafica4_Tension_Motores_Carro')
%Cerrar figura
close(figura)
msgbox('Graficas guardadas en "C:\PFC"','   ') %Muestra un mensaje para
que el usuario sepa donde se ha guardado.
```

•Obtener variables a partir de un archivo .fig.

Abrimos la figura y en Matlab escribimos los siguientes comandos:

Cuando contiene una sola gráfica:
lh = findall(gca, 'type', 'line');
xx = get(lh,'xdata');
yy = get(lh,'ydata');
Variable_X=xx'
Variable_Y=yy'

Cuando contiene más de una gráfica:

lh = findall(gca, 'type', 'line');
xx = get(lh,'xdata');
yy = get(lh,'ydata');
Variable_X1=xx{1}'
Variable_Y1=yy{1}'
Variable_X2=xx{2}'
Variable_Y2=yy{2}'

Si se desea guardar las variables en un archivo de texto utilizaremos el siguiente comando:
save 'Nombre_Archivo.txt' Variable_a_Guardar -ascii

Anexo C. Problemas encontrados en el desarrollo del proyecto.

C.1. Enviar valores para el PID del carro.

Este problema ha sido importante ya que ha limitado la estructura de la GUI.
Como el NXTB del carro recibe información del NXTA y del PC a la vez, no podemos distinguir cual es el emisor. Si se quiere enviar los parámetros del PID se debe hacer en varios pasos. Primero se envía el P, luego el I y por último el D.
El inconveniente surge cuando el NXTA envía el valor del sensor de ultrasonidos mientras se está cambiando el PID, ya que se mezclarán los valores y saldrá un PID no deseado.

El NXTA como solo recibe del PC no tiene ese problema.
A este problema no se ha encontrado solución.

C.2. Multiplicar los valores al enviar el PID por potencias de 10.

Otro inconveniente a la hora de enviar los valores es su formato. Los valores se envían en formato binario lo que limita el envío a 8 bits, es decir, valores comprendidos entre 0-255.
Como generalmente los valores de I y D son más pequeños que la unidad, se ha tomado esa medida para solucionar este problema, aunque si se quiere establecer un PID con exactitud se debe cambiar el PID desde RobotC.
Así, cuando se desee enviar el valor de I, este será multiplicado por 0.01 y cuando sea el valor de D, por 0.1.
Para entenderlo mejor, se puede observar la figura C.1

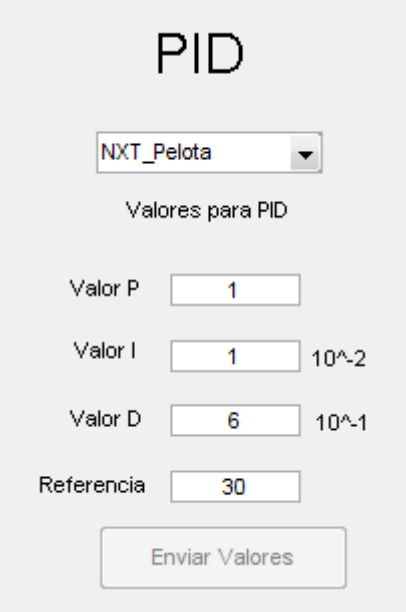


Figura AnexoC.1 Enviar Valores

C.3. Deslizamiento del carro al bajar por el tubo.

Cuando el carro y la pelota ya están estabilizados sobre la zona media del tubo.
Si la pelota baja bruscamente hacia una posición inferior, el carro baja por el tubo buscando la pelota lo que produce un deslizamiento y así un error de medida en el encoder del motor.

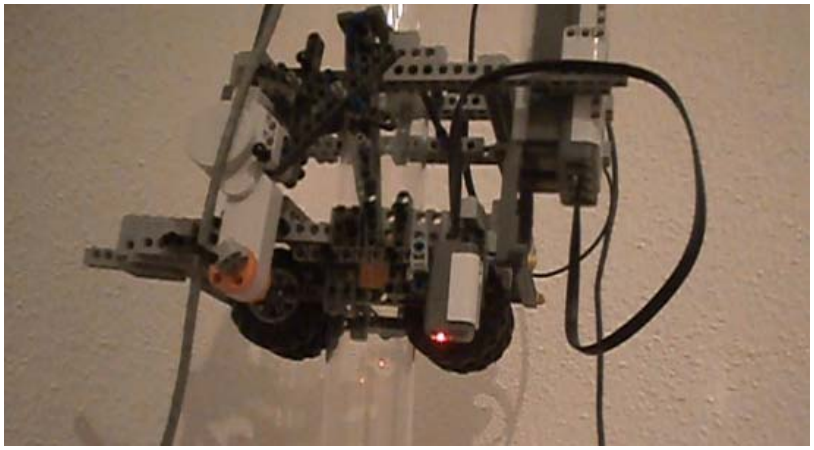


Figura AnexoC.2 Carro con error en el encoder

Una solución aceptable aunque no final fue utilizar el sensor de luz contenido en el carro y que no tenía función alguna.

Se cambió el sensor de luz de sitio para que cuando el carro estuviera en la posición 0, es decir, las ruedas tocaran en la mesa de madera, el sensor de luz reseteara el encoder y así solucionar este error de medida.

El inconveniente que tiene esta solución, es que si el carro no llega hasta la mesa, el sensor de luz no actúa y se tendría este error durante nuestra prueba.

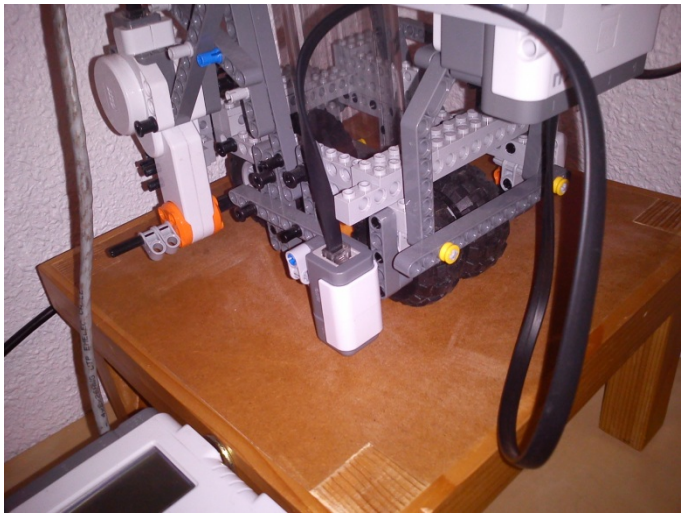


Figura AnexoC.3 Sensor de luz.

C.4 GUI. Valores de variables en tiempo real.

En la GUI cada elemento contiene su código de ejecución que se activa cuando interactuamos con él.
El problema surge cuando se quiere cambiar la referencia en la gráfica 1 mientras se envían los valores del PID de la pelota.
Como el código del botón Comenzar está en ejecución, se debe obtener el valor de la referencia desde el Handles.
El inconveniente viene en la parte de código del botón Enviar valores, ya que la GUI solo permite guardar una vez el valor en el Handles, ya que si se vuelve a ejecutar otra vez el mismo botón, la GUI se bloqueará.

Se optó por crear un Static Text en el cual se asigna el valor de la referencia y el botón Comenzar obtendrá ese valor en tiempo real.

En el caso del botón Establecer tiempo pasa lo mismo ya que si se quiere cambiar el tiempo al hacer una segunda prueba, daría problemas por lo que también se incluyó otro Static Text.

Como es normal, estos botones se pusieron invisibles para ya que no son necesarios mostrarlos.

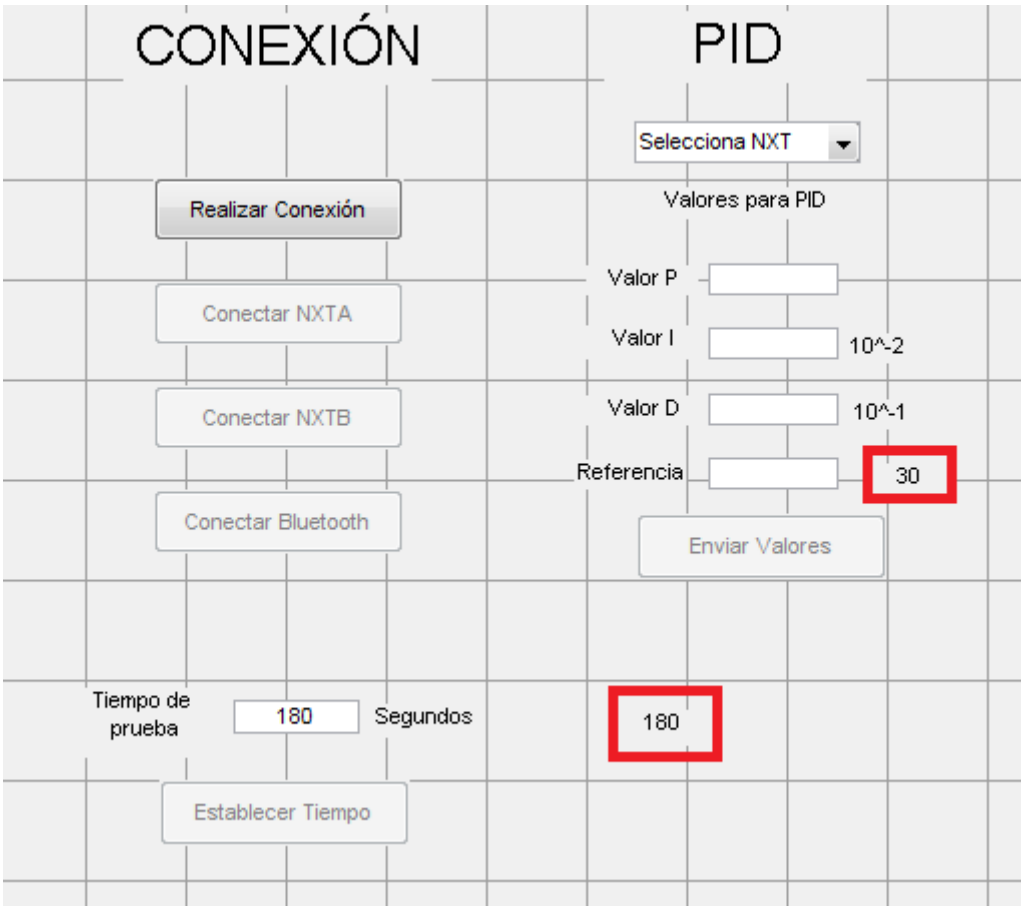


Figura AnexoC.4 GUI sin ejecutar.

C.5. Leyendas en graficas GUI.

Las gráficas Axes mientras se está ejecutando la GUI, se actualizan por puntos, y en cada punto que añade, vuelve a añadir el título, nombre de los ejes y la leyenda.
El problema aparece solo en la leyenda ya que al estar justamente dentro del cuadrado blanco de la gráfica, que es donde se aprecia el error, cada vez que se actualiza un punto, también se ve como se actualiza la leyenda.

Por lo que se veía como parpadeaba la leyenda mientras se ejecutaba la GUI.
Para solucionarlo se optó por poner la leyenda al acabar la gráfica así no se apreciaría este error.