ISSN: 1692-7257 - Volumen 2 – Número 28 - 2016
Revista Colombiana de Tecnologías de Avanzada

Recibido: 25 de enero de 2016
Aceptado: 4 de marzo de 2016

HERRAMIENTA DE SOFTWARE PARA COMANDAR CONTROLADORES DE POSICION DESDE MATLAB

COMPUTER TOOLBOX FOR SENDING COMMANDS TO POSITION CONTROLLER USING MATLAB

PhD. Eugenio Yime Rodriguez*, PhD. Javier Roldán Mckinley**
PhD. Pedro F. Cárdenas***

*** Universidad Tecnológica de Bolívar**, Ingeniería Mecatrónica, Grupo GIMAT.
Km. 1 Vía a Turbaco, Parque Tecnológico Carlos Vélez Pombo, Cartagena de Indias.
E-mail: eyime@unitecnologica.edu.co.
**** Universidad del Atlántico**, Ingeniería Mecánica, Grupo DIMER.
Km 7 Antigua Vía a Puerto Colombia, Piso 4, Bloque H, Barranquilla, Atlántico.
E-mail: javierroldan@mail.uniatlantico.edu.co.
***** Universidad Nacional**, Ingeniería Mecatrónica, Grupo UN Robot.
Cr 30 # 45-03, Bogotá, Colombia.
E-mail: pfcardenash@unal.edu.co.

Resumen: Se resume el diseño y desarrollo de una herramienta computacional que facilita el control y uso de los controladores de EPOS2 de Maxon Motors desde Matlab y Simulink. El desarrollo de la herramienta apunta inicialmente al campo académico e investigativo de la Robótica, por cuanto al facilitar el uso del control de los motores les permitirá a los diseñadores de sistemas robóticos enfocarse en la cinemática y diseño de interfaz gráfica de usuario de sus aplicaciones.

Palabras clave: Maxon Motors, Motores EPOS2, Controladores de Motores DC.

Abstract: It is summarized the design and development of a software toolbox for Matlab and Simulink created to assist in the use and control of Maxon Motors EPOS2 controllers. The toolbox initially targets the Robotics academic and research area, since any help provided with the actuators control will let the Robotics designer focusing on the kinematics and the graphical user interface in their applications.

Keywords: Maxon Motors, EPOS2 Motors, DC Motor Controllers.

1. INTRODUCCIÓN

Los controladores y motores Maxon Motors son ampliamente reconocidos en el área de Robótica por su excelente rendimiento y facilidad de uso. Alguna de las aplicaciones robóticas que hacen uso de motores y controladores Maxon son, el Mars Sojourner (J. M., 1997), Mars Oportunity, (Eisen et al., 1997), la futura misión ExoMars, (Phillips et al., 2012), la mano DLR-HIT, (Liu et al., 2007), el robot ayudante de cirugía da Vinci S HD, (Maxon Motors, 2011A), (Maxon Motors, 2011B), (Torres et al., 2013), entre otros.

En robótica, los actuadores son el músculo del robot, por lo que una buena selección de los motores en conjunto con el sistema de control de posición de los mismos es un paso fundamental que determina el grado de éxito de la aplicación Robótica. En el caso de los controladores de posición EPOS2 de Maxon Motors, estos sobresalen por su versatilidad de uso, ya que

cuentan con diversas interfaces que le permiten que se utilicen con las entradas y salidas digitales, las entradas análogas, con los protocolos digitales USB y RS-232, o el protocolo CAN.

La comunicación nativa de los controladores EPOS2 se realiza a través de una red CAN, por cuanto son controladores que cumplen con las normas CiA, Can in Automation, DS-301, DSP-305 y DSP-402. En el trabajo propuesto por Yime et al. (2008) se ilustra una forma de utilizar dichos protocolos para controlar un robot Stewart-Gough empleando controladores EPOS 24/1.

El protocolo RS-232 es un protocolo digital ampliamente conocido y utilizado en robótica, el cual se puede emplear en aplicaciones que requieran tiempo real. Sin embargo, en los últimos años se ha desplazado su uso por el bus USB debido, en parte, a que los nuevos computadores y portátiles no suelen traer este puerto. Para los controladores EPOS, la versión previa a los EPOS2, existe una librería *open-source* que permite utilizarlos desde aplicaciones escritas en C, no disponible en Matlab, (Hauser, 2014). Para el caso de los EPOS2, (Caruis, 2014) ilustra una forma de comunicarse desde Matlab con el protocolo RS-232, mediante lenguaje nativo escrito en archivos con extensión ".m". En cuanto al uso del protocolo USB, a pesar de no ser un protocolo de comunicación en tiempo real, suele ser empleado en diseños robóticos académicos por su facilidad de uso.

El presente artículo se enfoca en el diseño y creación de una herramienta para Matlab/Simulink que permita usar los controladores EPOS2 con puerto USB. Esta herramienta se diferencia de la propuesta por Hauser (2014) porque se implementan todos los modos de operación comunes en los controladores EPOS2. También se distingue de Carius (2014) en que el código de comunicación se escribe en C para compilarse a archivos ".mex", lo cual acelera un poco la ejecución desde Matlab (Rodríguez O. et al., 2012).

2. CONTROLADORES EPOS2

Los controladores EPOS2 son controladores de motores DC con escobillas (Brushed) o sin escobillas (Brushless) que se caracterizan por su versatilidad. Cuentan con diferentes protocolos de comunicación, ya sean digitales o análogos que los permiten emplear en diferentes condiciones de uso. La figura 1, ilustra los tres tipos de controladores EPOS2 24/2 que vende Maxon Motors. La figura 1a muestra un controlador de motores DC con escobillas; para la figura 1b se tiene un controlador de motores DC sin escobillas, y en la figura 1c un controlador híbrido para motores con y sin escobillas (Moreno et al., 2013).

Los EPOS2 están basados en modos de operación los cuales determinan la forma como el controlador opera. Los modos de operación más comunes son: HOME, para hacer el cero del eje; POSITION, para controlar la posición del motor basado en un controlador PID; PROFILE POSITION, para controlar la posición según un perfil de aceleración trapezoidal o senoidal; VELCITY, para controlar la velocidad en rpm del motor usando un controlador PID; PROFILE VELOCITY, para controlar la velocidad según los perfiles de aceleración antes mencionado; y CURRENT, para controlar la corriente que circula por el motor según un controlador PID.

Fig. 1a. Controlador de motores con escobillas

Fig. 1b. Controlador de motores sin escobillas

Fig. 1c. Controlador híbrido

Fig. 1. Tipos de controladores EPOS2 24/2.

Los EPOS2 tienen otros modos de operación como el MASTER ENCODER MODE, que permite operarlo como un engrane electrónico y STEP/DIRECTION MODE, que hace que se puedan utilizar como un motor paso a paso. Sin embargo, estos dos últimos modos no suelen emplearse en una aplicación robótica.

Si los controladores EPOS2 se emplean para trabajar con comunicación análoga, tal como lo hace un servoamplificador clásico, se debe configurar en el entorno gráfico, EPOS Studio, para que se comporte en uno de los modos de operación antes mencionado y proceda a leer una de las entradas análogas, la cual usualmente está configurada en el rango de ±10V, para determinar el comando que a seguir el motor.

En el caso de comunicación digital, que puede ser RS-232, USB o CAN, se tiene mayor libertad, pues se puede cambiar entre los diferentes modos de operación, leer el estado, resetear los errores, enviar comandos, cambiar la configuración de los PID, de los perfiles, etc.

En lo que respecta al desarrollo realizado, la comunicación se basará en USB. Si bien las posibilidades de uso de los motores son muy diversas, en una aplicación robótica se suele seguir un patrón convencional: abrir el puerto, conectarse con el nodo, verificar errores, borrar errores en caso de que el nodo este en error, cambiar a uno de los modos de operación, habilitar el nodo, enviar comandos según el modo de operación, lo cual se repite hasta que se finalice el uso del motor, tras lo cual se deshabilita el motor, y se procede a cerrar la comunicación. La figura 2 ilustra este proceso convencional de trabajar con un controlador EPOS2.

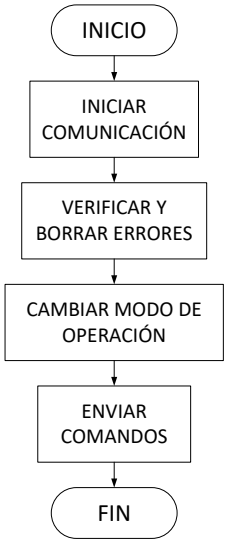

Fig. 2. Forma convencional de interactuar con un controlador EPOS2 24/2.

3. DESCRIPCIÓN DE LA HERRAMIENTA

La herramienta computacional para la comunicación USB con los motores EPOS2 24/2 se construyó en C y se compiló con MATLAB utilizando la instrucción "mex". Esto generó unos ejecutables en MATLAB, los cuales garantizan un tiempo de ejecución menor que el tiempo requerido en la ejecución de las mismas sentencias escritas en el lenguaje nativo de MATLAB. Se programaron los seis modos de operación más utilizados: *Position*, *Profile Position*, *Velocity*, *Profile Velocity*, *Current* y *Homming.*

La herramienta consiste de 25 funciones escritas en C y compiladas para MATLAB como. mex, con sus respectivos archivos .m de ayuda para visualizarse al llamar el comando help. Además cuenta con 6 archivos adicionales .m para la instalación, limpieza, ejemplos de uso y las clases para programación objeto escritas en.m.

A continuación, se hace una breve descripción de los comandos programados en C según el modo de operación.

3.1. Apertura de la comunicación

El primer paso antes de utilizar la herramienta es abrir un lazo de comunicación con el nodo. Esto se hace llamando a la instrucción "**OpenCommunication**", la cual no toma parámetros y devuelve un entero relacionado con el manejador de Windows del puerto USB.

3.2. Verificación de errores del nodo

Una vez se establece la comunicación, se procede a verificar si el nodo se encuentra en error, en caso de tenerlo se debe borrar el error. Las funciones que hacen esto son: "**GetErrorState**" y "**ClearErrorState**". Ambas funciones toman como parámetros el manejador del puerto y el número del nodo. Ejemplo:

```
>> % Abrir el puerto USB
>> a = OpenCommunication;
>> % Borrar el error del nodo 1
>> ClearErrorState(a, 1);
```

3.3. Cambiar el modo de operación

Antes de enviar comandos al nodo EPOS2 se debe cambiar el modo de operación para que entienda los comandos que se desean. Se llama a la función "**SetOperationMode**" para cambiar entre los diferentes modos. Esta función toma como parámetros el manejador USB, el número del nodo

y un entero simbolizando el modo deseado. Los diferentes modos de operación se muestran a continuación.

```
>> % Cambiar modo de operación
>> % 1 para HOMMING
>> % 2 para CURRENT
>> % 3 para VELOCITY
>> % 4 para POSITION
>> % 5 para PROFILE VELOCITY
>> % 6 para PROFILE POSITION
>> % Ejemplo para nodo 1 en modo CURRENT
>> SetOperationMode(a, 1, 2);
```

De igual forma, se creó la función "**GetOperationMode**" para saber el modo actual de operación del nodo. La función devuelve un entero en el rango de 1 a 6, con el mismo significado mencionado para "**SetOperationMode**".

Un ejemplo de su uso se presenta a continuación.

```
>> % Ejemplo de GetOperationMode
>> b = GetOperatonMode(a, 1);
```

3.4. Habilitar el nodo

Antes de enviar comandos al nodo EPOS2, es necesario habilitar la etapa de potencia, esto se hace llamando a la función "**EnableNode**". De igual forma, para deshabilitar un nodo se llama a "**DisableNode**". Ejemplos de uso:

```
>> % habilitar un nodo
>> EnableNode(a, 1);
>> % deshabilitar un nodo
>> DisableNode(a, 1);
```

3.5. Comando según el modo de operación

Para cada modo de operación se crearon una serie de comandos que permiten interactuar con el nodo. En las siguientes subsecciones se explican los comandos desarrollados.

3.5.1. Modo Homming

El modo de operación HOMMING permite hacer el cero máquina del robot, al hacer que cada eje vaya a su punto cero físico y a partir de ahí comience la medición de los desplazamientos. En la herramienta se permite trabajar con el nodo EPOS2 al cual se le puede conectar un interruptor para hacer el cero por detección del mismo, o por sobrepaso de corriente. La función que se creó en MATLAB es "**FindHome**", la cual acepta un entero de 1 a 8 que simboliza el tipo de modo cero que se quiere realizar.

En el caso de 1 se realiza un home poniendo en cero la posición actual del encoder. Para el valor de 2 se realiza un home moviendo el motor con velocidad negativa hasta que toque el interruptor de cero. Para 3 se realiza igual que 2 excepto que se realiza una búsqueda a cero del canal índice del encoder.

Para el home tipo 4 se realiza una búsqueda a cero con velocidad positiva del motor hasta que toque el interruptor de cero. Para el caso de 5, se hace igual que 4 pero se agrega la búsqueda a cero del canal índice del encoder.

Para el valor de 5, se hace una búsqueda a cero con velocidad negativa hasta que el eje encuentre un tope mecánico que hace que la corriente suba. En el caso de 6, se repite la búsqueda por corriente, pero se le agrega el cero del índice del motor. Con el valor 7, se hace la búsqueda con corriente, pero con velocidad positiva del motor.

Por último, para el valor de 8, se procede a hacer la búsqueda del cero con velocidad positiva agregando la búsqueda a cero del índice. Ejemplo de uso de "**FindHome**":

```
>> % FindHome en el nodo 1 a la posición actual
>> FindHome(a, 1, 1);
```

3.5.2. Modo Current

En el modo current se establece la corriente que debe pasar por el motor, las funciones que acepta la herramienta en este modo son "**SetCurrent**" y "**GetCurrent**". La función "**SetCurrent**" es para enviar el valor deseado de corriente, mientras que "**GetCurrent**" devuelve el valor actual de corriente que circular por el motor. Ejemplo de uso:

```
>> % valor deseado de 500 mA en el motor 1
>> SetCurrent(a, 1, 500);
>> % Valor actual en el motor 1
>> b = GetCurrent(a, 1);
```

3.5.3. Modo Velocity

En el modo velocidad el nodo EPOS2 24/2 actúa como un PID que controla la velocidad del motor. El PID del motor debe sintonizarse cuando se configura el nodo desde el software EPOS STUDIO. Las funciones programadas fueron "**SetVelocity**" para cambiar el valor deseado de velocidad y "**GetVelocity**" para obtener el valor actual de velocidad. Ejemplo:

```
>> % cambio de la velocidad a 500 rpm
>> SetVelocity(a, 1, 500);
>> % obtener velocidad actual
>> b = GetVelocity(a, 1);
```

3.5.4. Modo Position

Este modo, al igual que los dos anteriores, hace que el nodo actúe como un PID controlando el motor. En este caso, se controla la posición. Las funciones programadas fueron: "**SetPosition**" para enviar el valor deseado de posición y "**GetPosition**" para obtener el valor actual del motor. Ejemplos:

```
>> % Mover a 150000 pulsos de encoder
>> SetPosition(a, 1, 150000);
>> % Obtener posición actual
>> b = GetPosition(a, 1);
```

3.5.5. Modo Profile Velocity

En este modo el sistema mueve el motor hasta una velocidad deseada utilizando un perfil de aceleración trapezoidal o sinuidal.

En este modo se programaron las funciones "**SetProfileVelocityData**" y "**MoveWithVelocity**". La primera ajusta el valor de aceleración/deceleración del perfil trapezoidal. La segunda establece el valor deseado de rpm en el motor. Ejemplo:

```
>> % ajustar aceleración a 5000 rpm/s
>> SetProfileVelocityData(a, 1, 5000);
>> % mover a 1000 rpm
>> MoveWithVelocity(a, 1, 1000);
```

Para saber la velocidad actual del motor, se llama a la función "**GetVelocity**", al igual que en caso del modo velocity.

3.5.6. Modo Profile Position

Este modo consiste en mover el motor de un punto a otro utilizando un perfil de aceleración trapezoidal o senoidal. Las funciones que se programaron fueron: "**SetProfilePositionData**" y "**MoveToPosition**". La primera función se encarga de ajustar los parámetros del perfil de aceleración trapezoidal, los valores de entrada son la velocidad y aceleración del perfil. La segunda función hace que el motor se mueva el valor deseado siguiendo el perfil.

Ejemplo:

```
>> % perfil de 3000 rpm y 2000 rpm/s
>> SetProfilePositionData(a, 1, 3000, 2000);
>> % mover a 150000 pulsos de manera relativa
>> MoveToPosition(a, 1, 150000, 0);
>> % mover a -100000 pulsos absolutos
>> MoveToPosition(a, 1, -100000, 1);
```

3.5.7. Otras funciones programadas

Adicional a las funciones antes mencionadas, se programaron: "**QuickStop**", para detener el motor; "**IsTargetReached**", para saber si el motor alcanzó el estado estable para el comando que fue enviado.

Esta función es útil en modo posición, ya sea perfil posición o solo posición, por cuanto ayuda a saber si el motor logró la posición deseada. "**WaitForTargetReached**", sirve para esperar hasta que el motor logre el objetivo trazado. "**SetObject**" se creó para cambiar el valor de un objeto en el diccionario de objetos del nodo EPOS2; y "**GetObject**" para obtener el valor actual del objeto.

Algunos ejemplos de uso son:

```
>> % Detener el nodo 1
>> QuickStop(a, 1);
>> % Esperar a que el motor alcance la posición
>> MoveToPosition(a, 1, 100000, 0);
>> WaitForTargetReached(a, 1);
```

3.5.8. Finalizar con el nodo

Por último, cuando ya se terminaron de enviar los comandos al nodo, se procede a deshabilitarlo y terminar con la comunicación. Para ello se llama a "**DisableNode**" y "**CloseCommunication**".

Ejemplo:

```
>> % Desabilitar el nodo
>> DisableNode(a, 1);
>> % cerrar comunicación
>> CloseCommunication(a);
```

4. PROGRAMACIÓN ORIENTADA A OBJETOS

Las funciones antes mencionadas se programaron en C, esto tiene la ventaja de mayor rapidez y posibilidad de uso en el entorno gráfico GUIDE. Sin embargo, para usuarios novatos con la forma de uso de los EPOS2 se creó una clase en MATLAB nativo, clase Epos2.m, la cual ayuda en la comunicación al tener programada una lista de chequeo de verificaciones y visualizar los posibles errores de lógica de los comandos. Por ejemplo, si el usuario manda un comando sin previamente habilitar el nodo, la clase visualiza dicho error, que se debe primero habilitar el nodo antes de enviar comandos.

En conjunto con la clase "Epos2" se crearon las clases "OperationModes" y "HommingMethods". Estas clases sirven de ayuda al momento de trabajar con un nodo ya que no hay que enviar números enteros sino una propiedad de dicha clase. Para establecer la comunicación con un nodo, el usuario hace un llamado al constructor de la clase epos2 con el número del nodo.

Por ejemplo, para establecer comunicación con el nodo 2 se llamaría mediante

```
>> Node2 = Epos2(2);
```

Para detectar si hay error, se llamaría al método "**IsInErrorState**", y para borrar el error se debe llamar al método "**ClearErrorState**". Se ha agregado un método llamado "**ExplainErrors**" cuya función es mostrar una explicación de los errores existente en el nodo EPOS2.
Ejemplo de cómo usar estas funciones serían

```
>> if (Node2.IsInErrorState)
>>     Node2.ExplainErrors;
>>     Node2.ClearErrorState;
>> end
```

Para cambiar los modos de operación se hace un llamado al método "**SetOperationMode**", el cual recibe como único parámetro una identificación de la clase "**OperationModes**". Ejemplo de uso.

```
>> % cambiar a modo home
>>Node2.SetOperationMode(OperationModes.
HommingMode );
>> % cambiar a modo corriente
>>Node2.SetOperationMode(OperationModes.
CurrentMode );
```

Si se va a cambiar al modo home, se puede hacer uso de dos métodos: "**SetHommingMethod**" y "**DoHomming**". El primero define el tipo de método a usar, el segundo realiza el procedimiento de home. El procedimiento a utilizar emplea la clase enumerada "**HommingMethods**", la cual contiene los ocho métodos posibles de hacer home. Por ejemplo, para hacer home a la posición actual se utilizaría:

```
>>Node2.SetHommingMethod(HommingMetho
ds.ActualPosition );
```

Una característica importante de la programación objeto es que se pueden recargar los métodos. En la clase "**epos2**" se ha recargado "**SetHommingMethod**" para que acepte varios parámetros dependiendo del tipo de home a seguir. En los procedimientos que se realiza una búsqueda del switch de home se puede especificar la velocidad y aceleración de la búsqueda. En aquellos donde se realiza una búsqueda del índice, se puede agregar la velocidad de búsqueda del cero y la velocidad y aceleración de la búsqueda del índice. Ejemplos:

```
>> % cambiar a vel positiva sin índice
>> % velocidad de búsqueda de 10 rpm
>> % aceleracion de 5 rpm/s
>>Node2.SetHommingMethod(HommingMethods.
HomeSwitchPosSpeed, 10, 5);
>> % hacer el home
>> Node2.DoHomming;

>> % cambiar a vel negativa e índice
>> % velocidad de búsqueda del cero a 200 rpm
>> % velocidad de búsqueda del índice a 20 rpm
>> % aceleracion en ambos casos de 1000 rpm/s
>>Node2.SetHommingMethod(HommingMethods.
HomeSwitchNegSpeedIndex, 200, 20, 1000 );
>> % hacer el home
>> Node2.DoHomming;
```

Para enviar comandos en el modo posición y perfil posición, se utiliza el método "**MotionInPosition**" el cual se encuentra sobrecargado y puede aceptar dos variantes: una con el valor deseado de la posición del encoder y otra con el valor deseado posición más la velocidad del perfil, la aceleración del perfil y si la posición es absoluta o relativa. Ejemplos de uso.

```
>> % mover a 150000 pulsos de encoder
>> Node2.MotionInPosition(150000);

>>% mover a 10000 pulsos de manera absoluta
>>% con velocidad de 100 rpm y aceleracion de
>>% 200 rpm/s
>> Node2.MotionInPosition(10000, 100, 200, 1);
```

En el caso de los modos velocidad y perfil velocidad, se utiliza el método "**MotionInVelocity**", el cual también se encuentra sobrecargado y tiene dos variantes, una con la velocidad deseada y otra con las especificaciones de la aceleración del perfil.

Ejemplos de uso:

```
>> % mover a 2000 rpm
>> Node2.MotionInVelocity(2000);

>>% mover a 2000 rpm
>>% con aceleracion de 1000 rpm/s
>> Node2.MotionInVelocity(2000, 1000);
```

Para el modo corriente, la clase tiene el método "**MotionWithCurrent**", el cual no se encuentra sobrecargado y solo acepta un único parámetro: la corriente deseada del motor.

```
>> % motor con 2 amperios
>> Node2.MotionWithCurrent( 2000 );
```

Las ventajas de usar la clase "Epos2" en vez de la programación procedural se relacionan con la validación de las tareas antes de ejecutar una acción.

Así por ejemplo, antes de enviar cualquiera de los comandos "*Motion.*." se hace la validación del nodo y se asegura que no tenga errores, y que el mismo se encuentre habilitado, en caso contrario se le indica al usuario que haga las correcciones del caso.

Otra de las validaciones que se hace es que se encuentre en el modo correcto, por ejemplo, para "*MotionInVelocity*" se debe estar en el modo velocidad o perfil velocidad. También se validan que la función sobrecargada "*MotionInVelocity*" con tres parámetros se utilice únicamente en el modo perfil velocidad.

5. USO EN SIMULINK

La herramienta desarrollada cuenta con una *s-function* que permite utilizar los nodos EPOS2 en Simulink. La *s-function* se escribió directamente en C para hacer que su operación sea computacionalmente más eficiente. Para su ejecución, el usuario solo debe ingresar tres parámetros: el número del nodo, el modo de operación y el tiempo de muestreo deseado. La figura 2 ilustra un ejemplo de un modelo con la *s-function* incluida. Si bien la entrada de la *s-function* es programable, la salida será siempre la posición actual del motor.

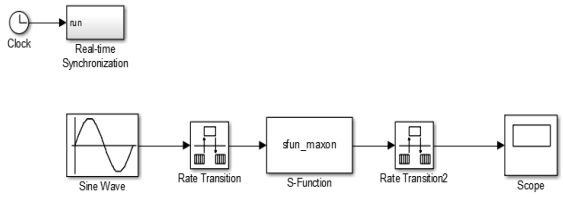

Fig. 2. Ejemplo básico de uso en Simulink.

La figura 3 ilustra los parámetros configurables de la *s-function*. Los modos de operación aceptados son: 1 para corriente, 2 para velocidad y 3 para posición.

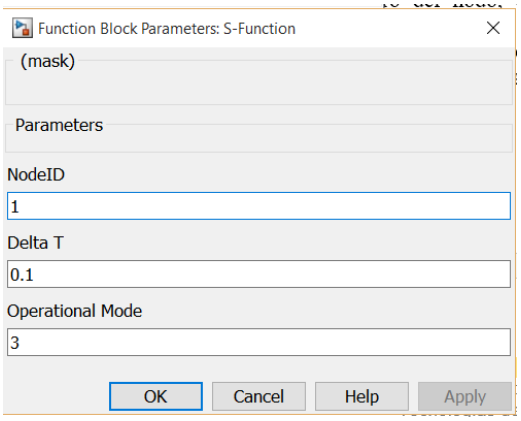

Fig. 3. Ejemplo de parámetros de la s-function.

En el ejemplo de la figura 2, el cual se puede descargar desde la página (Yime, 2015), se desea un movimiento senoidal en posición del motor. La salida será el mismo movimiento, puesto que el PID interno del EPOS2 hace que se cumpla con los parámetros deseados, tal como se ilustra en la figura 4. La única diferencia será el retardo en la comunicación debido al tiempo de muestreo de la *s-function*. En la figura 4 se puede apreciar que este retardo hace que la salida se vea de manera escalonada.

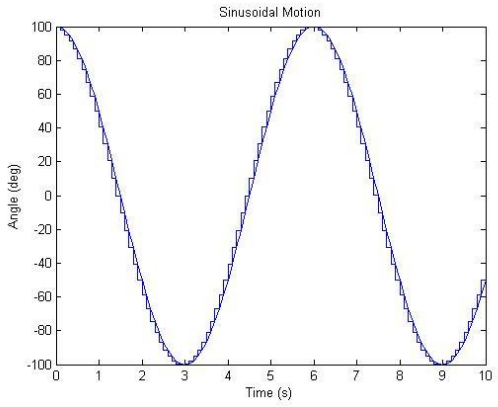

Fig. 4. Gráfica del control de posición con entrada senoidal.

6. FOMENTO DE LA HERRAMIENTA EN APLICACIONES ROBÓTICAS

Para fomentar el uso de la herramienta en aplicaciones robóticas, esta se ha compartido a forma de código *open-source* en la web de *Mathworks* denominada *Matlab File Exchange*, (Yime, 2015) y se puede localizar fácilmente desde un navegador buscando el título "*Commanding Maxon Motors EPOS2 Motor Controller from MATLAB*".

7. CONCLUSIONES

Se ilustró el diseño y desarrollo de una herramienta computacional para trabajar con los controladores de motores EPOS2 24/2 desde Matlab/Simulink. A diferencia de otros códigos similares, la herramienta está enfocada facilitar el uso por el usuario ya sea a través de funciones en una programación procedural o a través de objetos en una programación orientada a objetos. Con el desarrollo de la herramienta se busca incentivar a los estudiantes de pregrado a realizar proyectos académicos robóticos facilitándole el uso de los controladores al tener como ayuda el entorno de desarrollo ofrecido por Matlab/Simulink. Desde el punto de vista informático, la herramienta desarrollada le permite al usuario ejecutar funciones desarrolladas en lenguajes de programación de alto nivel y ejecutarlas en ambiente gráfico de bloques (Simulink), o en programas de entorno conocido tal como los scripts ".m" de Matlab.

8. AGRADECIMIENTOS

Este trabajo fue posible gracias al apoyo económico del Departamento de Investigaciones de la Universidad Tecnológica de Bolívar que en conjunto con COLCIENCIAS hicieron posible la comprar de los equipos utilizados. COLCIENCIAS también apoyó con la financiación de la Beca de Doctorado del ingeniero Pedro Cardenas Hernández.

REFERENCIAS

Carius J. (2014). Dynamic Maneuvers with a Single-Wheel Robot. Bachelor's thesis, Swiss Federal Institute of Technology. Zurich.

Rodríguez Oscar Oswaldo, Pineda Pinto Ronald Fernando, Cárdenas Pedro Fabián. (2012). herramientas EJS 3D/MATLAB para el control del sistema no lineal aplicado al péndulo invertido sobre carro deslizante. Revista colombiana de tecnologías de Avanzada. 1 (19). Pág. 28 – 34.

Eisen, H. J., Buck, C. W., Gillis-Smith, G. R., Umland, J. W. (1997). "Mechanical design of the mars path_nder misión". In Proceedings of the 7th conference on European Space Mechanisms and Tribology Symposium, pages 293-301.

Hauser, M. (2014). Librería para control de motores EPO. libepos - a c library to control the epos motor controller. http://sourceforge.net/projects/libepos/. (Consultado: 1 de octubre de 2015)

Moreno Rubio J., Jiménez López A, Barrera Lombana N. (2013). El amplificador de potencia de carga sintonizada. Revista colombiana de tecnologías de Avanzada. 2(22). Pág. 9 – 13.

Liu, H., Meusel, P., Seitz, N., Willberg, B., Hirzinger, G., Jin, M., Liu, Y., Wei, R., and Xie, Z. (2007). "The Modular Multisensory dlr Hithand". Mechanism and Machine Theory, Vol. 42, No. 5.

M., J. (1997). "Maxon Motors Powers Rover on Pathfinder Mission". Military and Aerospace Electronics, Vol. 8, No. 10(27).

Maxon Motors (2011a). Robots para operaciones mínimamente invasivas.
http://www.maxonmotor.es/medias/sysmaster/8811798331422/2011-01-es-surgical-robots.pdf?attachment=true.
(Consultado: 1 de octubre de 2015)

Maxon Motors (2011b). Robots quirúrgicos para intervenciones mínimamente invasivas.
http://www.maxonmotor.es/maxon/view/application/SURGICALROBOTS-AB.
(Consultado: 1 de octubre de 2015)

Torres Clayton José, Archila John Faber, Tronco Mário Luiz, Becker Marcelo, Viera Porto Arthur José, Tiberti Alexander José. (2013). Estudio cinemático de una plataforma robótica para agricultura. Revista colombiana de tecnologías de Avanzada. 2 (22). Pág. 131 – 137.

Phillips, R., Palladino, M., and Courtois, C. (2012). "Development of Brushed and Brushless DC Motors for use in the Exomars Drilling and Sampling Mechanism". In Proceedings of the 41st Aerospace Mechanisms Symposium.

Yime, E., Quintero, J., and Saltaren, R. (2008). "CAN on Parallel Robots: How to Control a Stewart Platform using CAN Based Motor Controllers". In Proceedings of the 12th International CAN Conference (ICC).

Yime, E. (2015). Commanding Maxon Motors EPOS2 Motor Controllers from MATLAB. http://www.mathworks.com/matlabcentral/fileexchange/53735-commanding-maxon-motors-epos2-motor-controller-from-matlab
(Consultado: 1 de octubre de 2015)