Artículo **Revista del Desarrollo Tecnológico**
Marzo 2017 Vol.1 No.1 30-40

Laboratorio virtual de robótica basado en Matlab®/Simulink®/RobotStudio

LORETO-GÓMEZ, Gerardo*†, MORALES-MORFIN, Marcela, SÁNCHEZ-SUAREZ, Isahi' y VÁZQUEZ-INFANTE, J. Jesús

Instituto Tecnológico Superior de Uruapan
'Universidad Politécnica de Uruapan.

Recibido Enero 13, 2017; Aceptado Marzo 15, 2017

Resumen

Las herramientas de simulación han demostrado ser un mecanismo importante tanto para la enseñanza como para la investigación en el campo de la robótica industrial. El uso de estas herramientas proporciona múltiples funcionalidades no sólo en la teoría a nivel de conceptos, sino también a nivel de implementación. En este trabajo se propone un Laboratorio virtual de robótica basado en los programas de Matlab®/Simulink®/RobotStudio que tiene por objetivo vincular un sistema de programación real de robots industriales y un programa de simulación general que permita diseñar nuevos algoritmos de control. En el presente trabajo se desarrolló la aplicación de organización de objetos por color y forma que son detectados en el área de trabajo de un robot industrial a través de un sistema de visión integrado por el sensor Kinect. La información visual es enviada al simulador de ABB RobotStudio para genera automáticamente cada uno de los objetos en su entorno virtual, para finalmente, implementar la tarea de organización de los objetos por medio de un robot previamente configurado, lo anterior permite validar el potencial que tendría esta plataforma de simulación basada en los programas Matlab®/Simulink®/RobotStudio.

RobotStudio, Control visual, Robótica

Abstract

Simulation tools have proven to be an important mechanism for both teaching and research in the field of industrial robotics. The use of these tools provides multiple functionalities not only in theory at the concept level, but also at the implementation level. In this paper we propose a Virtual Robotics Laboratory based on Matlab®/Simulink® /RobotStudio programs that aim to link a real programming system of industrial robots and a general simulation program that allows the design of new control algorithms. In the present work it was developed the application of organization of objects by colour and shape that are detected in the work area of an industrial robot through a vision system integrated by a sensor Kinect. The visual information is sent to the simulator of ABB RobotStudio to automatically generate each of the objects in the virtual simulator, finally, implement the task of organizing these objects by means of a robot previously configured; this allows validating the potential that would have this simulation platform based on Matlab®/Simulink®/ RobotStudio programs.

RobotStudio, Visual control, Robotic

Citación: LORETO-GOMEZ, Gerardo, MORALES-MORFIN, Marcela, SANCHEZ-SUAREZ, Isahi y VAZQUEZ-INFANTE, J. Jesús. Laboratorio virtual de robótica basado en Matlab®/Simulink®/RobotStudio. Revista del desarrollo Tecnológico 2017, 1-1: 30-40

* Correspondencia al Autor (Correo Electrónico: gerardoloreto@tecuruapan.edu.mx)
† Investigador contribuyendo como primer autor.

Introducción

La simulación ha sido reconocida como una herramienta importante en el campo de la robótica industrial, diferentes herramientas son utilizadas para el análisis cinemático y dinámico de los robots industriales, para su programación fuera de línea, para el diseño de diferentes algoritmos de control, para el diseño de las estructuras mecánicas de los robots, para el diseño de celdas robóticas y líneas de producción, entre otras (Žlajpah, L., 2008), (Lázaro-Arvizu, Y., 2015).

La mayoría de los robots industriales disponibles comercialmente están soportados por sistemas de simulación especializados. El uso de herramientas de simulación profesional desarrolladas por los fabricantes de robots mezcla la riqueza de modelar con precisión los robots industriales físicos y los controladores reales. Por lo tanto, permiten trabajar con un modelo que está más cerca del comportamiento real del robot. Este es el caso del *Software* de simulación *RobotStudio*™ de ABB *Robotics* que es una herramienta que puede utilizarse con los robots disponibles de ABB y permite crear estaciones robóticas complejas al incorporar componentes y mecanismos creados por el usuario, un aspecto importante, es que permite obtener un ambiente de programación fuera de línea exacto a través de un controlador virtual que ejecuta el mismo código del controlador físico del robot (Connolly, C., 2009).

Por otro lado, existen diferentes herramientas de simulación que permiten el estudio de los temas relacionados con la Robótica. La librería denominada "*Robotics Toolbox for Matlab*" es tal vez una de las contribuciones más importantes en este ámbito desarrollada por Peter Corke (P. Corke, 1996). La librería se ejecuta bajo el entorno de Matlab® y ofrece un conjunto de herramientas muy completas que incluye una gran cantidad de algoritmos básicos y avanzados para el análisis de diferentes tipos de robots manipuladores.

Sin embargo, una de sus principales limitaciones desde el punto de vista para su aplicación con fines educativos con respecto a otras librerías, es que no tiene una representación realista de los robots y sólo utilizan una representación tipo alambre. Vila y Dominguez, analizan esta librería y sus limitaciones, presentando algunas sugerencias para hacerla más robusta y complementarla (Vila Rosado, D. N., & Dominguez López, J. A., 2005). Por otro lado Gil y colaboradores, desarrollan una librería denominada ARTE (*A Robotics Toolbox for Education*) basada en Matlab en la cual se pueden crear representaciones más realistas de los robots (Gil, Arturo, et. al, 2015). González y colaboradores, presentan el software de simulación SnAM, que consiste en un paquete de dominio público escrito en C++. La representación gráfica es manejada por bibliotecas OpenGL. Como resultado, SnAM proporciona una herramienta para simular con flexibilidad la cinemática de cualquier tipo de robot serial (González-Palacios, M. A., et. al, 2013).

Otra herramienta similar es presentada por López y colaboradores que permite simular la cinemática de varios robots industriales, desarrollada bajo Microsoft Visual C++ y que se denominada *RobotScene* (López-Nicolás, et. al., 2009). Damić y colaboradores, proponen un procedimiento para crear modelos visuales en 3D de robots y su exportación al programa *BondSim* (Damić, Vjekoslav; Carića, Ćira; Husić, Maida Ćohodar, 2014).

ISSN-2531-2960

El presente trabajo describe una aplicación de laboratorio virtual de robótica basada en el programa de Matlab® y *RobotStudio*™ que permite implementar un sistema de visión para la detección de objetos en el área de trabajo de un robot industrial, a partir de esta información se generan los elementos correspondientes en el simulador de *RobotStudio*™ y que son organizados por color y forma por el robot, lo anterior permite validar el potencial que tendría esta plataforma para la enseñanza de la teoría de robots industriales.

Descripción general de la plataforma

La plataforma está compuesta de tres elementos principales que son el sistema de visión, la estación de RobotStudio y finalmente el programa en Matlab, los cuales se describen a continuación:

Sistema de visión

En la robótica, la visión artificial es utilizada para que el robot pueda interactuar fácilmente con su entorno de trabajo. En el presente trabajo se utilizó el sensor Kinect que es un dispositivo desarrollado por la compañía Microsoft para la consola de videojuegos Xbox 360. El sensor Kinect contiene una cámara RGB (Rojo Verde Azul); un sensor de profundidad constituido por un emisor y una cámara de infrarrojos, ver figura 1.

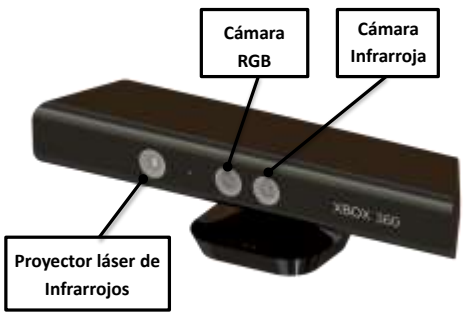

Figura 1 Microsoft Kinect.

Fuente: *Elaboración propia*

El sensor RGB es una cámara de video de 3 colores que se puede utilizar en dos configuraciones, la primera permite la captura de imágenes con una resolución de 640 x 480 pixeles a una velocidad de captura de 30 fotogramas por segundo, la segunda configuración obtiene imágenes de 1280 x 1024 pixeles a una velocidad de 15 fotogramas por segundo. El emisor de infrarrojos, emite una luz infrarroja que al pasar a través de un difusor genera un patrón de puntos que se proyectan sobre los objetos que se encuentren en el rango visible de 50 cm a 400 cm de profundidad. La cámara infrarroja, recibe la luz infrarroja proyectada y puede configurarse en tres modos; el primero, obtiene imágenes infrarrojas de 640 x 480 pixeles pero sin la profundidad de los objetos a una velocidad de 30 fotogramas por segundo; el segundo, tiene una resolución de 1280 x 1024 pixeles a una velocidad de 15 fotogramas por segundo; el tercer y último modo de funcionamiento adquiere tanto imágenes infrarrojas como de profundidad de 640 x 480 pixeles a una velocidad de 30 fotogramas por segundo.

La adquisición de imágenes se realiza en el entorno de Matlab; el primer controlador que se necesita instalar es el Kinect SDK (del inglés *Software Development Kit*) para Windows, en este caso se utiliza la versión 1.7 descargada de manera libre de la página de Microsoft. Una vez instalado el controlador, el LED del Kinect comienza a parpadear, esto significa que está correctamente instalado. Específicamente se requiere esta versión ya que versiones posteriores a la 1.8 no detecta los componentes del Kinect en Windows. A continuación, se debe instalar las librerías *Kinect for Windows Sensor*, además de la *OS Generic Video Interface*.

Para lograr la adquisición de imágenes en Matlab se utiliza la librería *Image Acquisition Toolbox* de Matlab. Antes de activar los sensores, se deben configurar los parámetros de captura de la imagen.

ISSN-2531-2960

La activación de los sensores del Kinect se hace mediante la función *start*. El sensor RGB del Kinect tiene como nombre predeterminado *Kinect 1* y el sensor de profundidad *Kinect 2*; para obtener una vista previa de las imágenes que está recibiendo el Kinect se usa la función *preview*. Una vez iniciados los sensores solamente se necesita capturar las imágenes que cada sensor está detectando, esto mediante la función *getsnapshot*.

El objetivo de la detección de objetos por el sistema de visión es el de simular el movimiento que deberá realizar un robot manipulador industrial para clasificar cada uno por color y por forma. Para eso es necesario conocer la posición y orientación en que se encuentra cada objeto, así como su altura, color y forma. Al realizar el procesamiento de la imagen, se obtiene los datos de posición en pixeles, por lo cual primero se realiza una calibración de tamaño entre pixeles y milímetros. Posteriormente, con la información visual obtenida se realizará la creación de los objetos virtuales en RobotStudio.

Para enviar la información de los objetos capturados por el sistema de visión a RobotStudio se debe construir un vector de información con la siguiente estructura: [IdentificadorID, TipoFigura, Color, Orientación, Posición]. El *IdentificadorID* es un dato que indica el tipo de información que se envia, para el caso de la información visual tendra un valor de cuatro. El *TipoFigura* representa dos posibilidades 1 para círculo y 2 para rectangulo. El *Color* correspondera a 1 para rojo, 2 para verde y 3 para azul. La *Orientación* se indicara con respecto a los angulos de Euler y la *Posición* correspondera a las coordenadas X,Y,Z. Para el caso de la coordenada Z se consideran dos posibilidades, la primera considera una altura constante para todos los objetos y el segundo caso, consiste en enviar la altura obtenida por el sensor Kinect.

RobotStudio interpreta este vector y dibuja los objetos en el entorno de trabajo, sin embargo, presenta la desventaja de que, al extrudir los objetos con la altura obtenida por el sensor Kinect no es posible representar el color en el simulador, por lo que todas las figuras quedan construidas de color gris, sin embargo, el robot las clasifica de acuerdo a los colores reales.

En la figura 2 se pueden observar las imágenes obtenidas con el comando *getsnapshot*, en este caso se procesa la imagen para identificar los rectángulos y los círculos de los colores rojo, azul oscuro y verde. Primeramente se realiza el aislamiento de cada color, se etiquetan los objetos y se procede a la identificación de rectángulos y círculos. Este procedimiento se aplica para cada color que se necesite identificar.

Figura 2 Objetos a clasificar, Izquierda: Captura RGB. Derecha: Captura Infrarrojos

Fuente: *Elaboración propia*

ISSN-2531-2960

El caso cuando se trabajar con el sensor de infrarrojos del Kinect para obtener la altura de los objetos, se procede a acceder a las coordenadas de los centroides de los objetos que hemos identificado en la imagen RGB, pero ahora en la imagen de profundidad. El valor que se encuentra en cada una de las celdas de la matriz que se genera a partir de la imagen de profundidad representa directamente la distancia en milímetros medida desde el plano del sensor hasta el plano del objeto deseado. La figura 3 muestra el resultado de la detección de objetos.

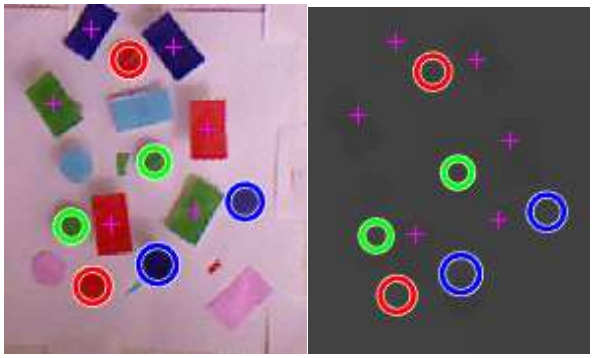

Figura 3 Objetos clasificados, Izquierda: Captura RGB. Derecha: Captura Infrarrojos

Fuente: *Elaboración propia*

Estación de RobotStudio

La estación robótica en RobotStudio debe poder realizar varias tareas para la aplicación a realizar, en este caso, desde la programación, el robot debe simular el movimiento de sus articulaciones a partir de los datos recibidos desde un software externo, que en este caso es Matlab, por lo cual, es requerido establecer la comunicación entre Matlab y RobotStudio. Además, la estación de trabajo tiene que crear los objetos capturados con la cámara en el ambiente de RobotStudio y crear la simulación de sujeción de los objetos a través de una herramienta con ventosa.

Primeramente, para lograr el envío y recepción de datos entre Matlab y RobotStudio se hace uso del protocolo de comunicación TCP/IP (del inglés *Transmission Control Protocol*/*Internet Protocol*). RobotStudio se programa como servidor para el envío y recepción de datos y Matlab será programado como el cliente (Frydrysiak M., 2014).

Para que se realice la comunicación para el envío y recepción de datos, se programan dos módulos en el código RAPID denominados *ConexiónTCPIP* e *InterpretacionDatos*. En el módulo de ConexionTCPIP se crean los elementos necesarios para establecer la conexión, a través del comando *SocketCreate* se crea un nuevo zócalo para comunicación. Después de haber creado el zócalo, se le asignan una dirección IP y un puerto mediante el comando *SocketBind*. Con estas instrucciones el zócalo queda configurado para ser utilizado como medio de comunicación con dispositivos externos al controlador del robot. Para lograr una correcta conexión se utiliza las funciones *SocketListen* y *SocketAccept*, estos comandos son ejecutados y pueden permanecer activos hasta que se detecte una solicitud de conexión externa. Una vez enlazados servidor y cliente, se utilizan los comandos *SocketReceive* y *SocketSend* para recibir y enviar datos respectivamente.

El código utilizado es el siguiente:

```
MODULE ConexionTCPIP

    !Definición de variables
  VAR socketdev ServidorRobotStudio;
  VAR socketdev ClienteMatLab;
  VAR bool      ConexionActual := FALSE;
  LOCAL VAR string    Ip:= "127.0.0.1";
  LOCAL VAR num       Puerto := 1024;

    !Procedimiento para la conexión TCP/IP
    PROC Conexion()
    SocketCreate ServidorRobotStudio;
```

```
    SocketBind ServidorRobotStudio, Ip, Puerto;
    SocketListen ServidorRobotStudio;
    SocketAccept ServidorRobotStudio, ClienteMatLab, \ClientAddress:=Ip, \Time:=WAIT_MAX;
    ConexionActual := TRUE;
  ENDPROC

ENDMODULE
```

En el módulo *InterpretacionDatos* a partir del primer elemento del vector de información enviado por el cliente y que ha sido denominado *IdentificadorID*, se seleccionará un caso para decodificar la información y poder ejecutar alguna de las tareas disponibles como son la creación de objetos (ID=4) a partir de la información proveniente del sistema de visión, la activación de ventosa (ID=3), el movimiento cartesiano (ID=2) o articular (ID=1) del robot.

La estructura que tiene el vector de información para realizar un movimiento de tipo articular es [IdentificadorID, SignoJunta_1, ValorJunta_1, …, SignoJunta_6, ValorJunta_6], los seis campos *ValorJunta* corresponden a los ángulos de las articulaciones expresados en grados y los seis campos *SignoJuntas* son utilizados para indicar los signos de cada valor articular, un cero identifica un dato negativo y uno identifica un dato positivo.

La estructura que tiene el vector de información para realizar un movimiento de tipo cartesiano es [IdentificadorID, Orientación, Posición], la información de la *Orientación* corresponde a tres datos con la siguiente estructura: [Signo, RZ, S, RY, S RX] y de la *Posición* tiene la siguiente estructura: [S, X, XR, S, Y, YR, S, Z, ZR]. La posición es dividida en dos datos: (X, Y, Z) contiene la información correspondiente a la cantidad de centenas de milímetros y (XR, YR, ZR) contiene la información de las unidades y decenas de milímetro.

Los seis campos (S) son utilizados para los signos de la orientación y de la posición (0 - negativo, 1 - positivo).

Por otro lado, los componentes inteligentes se utilizan dentro de la estación de trabajo de RobotStudio para simular un gran número de acciones y movimientos de los elementos en el entorno de trabajo que no son parte del robot, en el presente trabajo se utilizaron los componentes para simular la succión de una ventosas y otro para la creación de los objetos que son captados por el sistema de visión.

Para la creación de los objetos detectados por el sistema de visión es necesario crear un componente inteligente *Crear Objeto*. Este componente es capaz de crear un objeto de una cierta altura, color y forma. La creación de estos se logra a partir de la generación de copias de los mismos objetos que están prediseñados y ocultos en la estación de trabajo. En el caso de los objetos a una altura estándar, se crean tres cilindros, uno rojo, uno azul y uno verde, y tres tetraedros de los mismos colores, están ocultos y sus centroides en el plano XY se encuentran en la posición (0,0).

Cuando se reciben los datos de Matlab, el código RAPID se encarga de interpretarlos y enviar las instrucciones para copiar y posicionar los objetos. En el caso de los objetos de altura variable, se tienen dos formas de 1mm de altura, un cilindro y un tetraedro. Cuando se reciben las instrucciones, las formas son extruidas, una vez extruidas son copiadas, y las copias son manipuladas para posicionarse en el lugar deseado. La figura 4 muestra un ejemplo de una forma de cilindro construida como componente inteligente.

ISSN-2531-2960

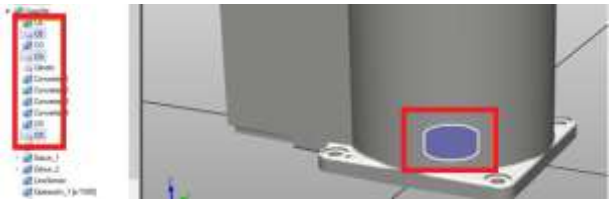

Figura 4 Componente inteligente objeto Cilindro

Fuente: *Elaboración propia*

En la figura 5 se muestran algunas de las señales de entrada y salida creadas para los componentes inteligentes de los objetos. Además de las señales digitales, también se utilizan señales analógicas, esto se debe a que se necesitan dimensionar los valores de las diferentes alturas y posiciones de los objetos.

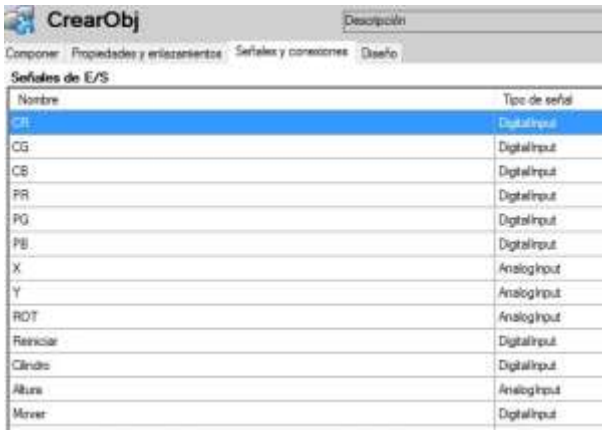

Figura 5 Señales para el componente inteligente crear objeto

Fuente: *Elaboración propia*

Finalmente, para que el robot pueda manipular los objetos se construye una herramienta que simula la sujeción por vacío a través de utilizar los componentes inteligentes de *LineSensor*, *Attacher* y *Detacher*, se agregan dos señales digitales de entrada, una para activar la succión y la otra para desactivarla.

Programa en Matlab

Para lograr el envío y recepción de datos entre MATLAB y RobotStudio inicialmente se crea un programa de inicio en el cual se realizara la activación del sensor Kinect y se establece la conexión, el código de instrucciones es el siguiente:

```
%% Iniciar Kinect
imaqreset
vid = videoinput('kinect',1);
vid2 = videoinput('kinect',2);
vid.FramesPerTrigger = 1;
vid2.FramesPerTrigger = 1;
vid.TriggerRepeat = 1;
vid2.TriggerRepeat = 1;
triggerconfig([vid vid2],'manual');
start([vid vid2]);

%%
robot=ControlRobot('127.0.0.1',1024);
robot.conectar;
robot.CI([180 0 180 300 1 350])
```

El código anterior hace uso de la clase *ControlRobot* que permiten generar la comunicación con RobotStudio (Gutiérrez Corbacho, A., 2014). La clase es creada con el fin de simplificar el uso de comandos para realizar la conexión y envió de información, realiza el cálculo de la cinemática inversa del robot a partir de los datos de las imágenes, y crea los valores de cada junta del robot, parte del código que permite la conexión es el siguiente:

```
classdef ControlRobot < handle
   %% Propiedades
   properties
      IP;
      puerto;
      conexion;
   end
   %% Métodos
   methods
      %% Función para Acceder a la
Dirección
      function r=ControlRobot(ip,puerto)
```

```
        r.IP=ip;
        r.puerto=puerto;
    end
    %% Funcion para Conectar con RobotStudio
    function conectar(r)
        r.conexion=tcpip(r.IP, r.puerto);
        fopen(r.conexion);
        P=uint8([1 1 0 1 0 1 0 1 0 1 90 1 0]);
        fwrite(r.conexion, P);
        out=fread(r.conexion,12);
        assignin('base','out',out);
    end …..
```

Mediante el comando *tcpip* se crea un objeto que representa al cliente de la red, y con el comando *fopen* se abre la comunicación, a partir de la utilización de estos comandos se puede manipular el envío y recepción de datos a través del puerto creado. Para escribir datos en el objeto creado se utiliza la función *fwrite*, y para leer los datos que existan se utiliza la función *fread*. Estas dos funciones permiten un envío y recepción de datos entre el servidor y el cliente. Los datos que se utilizan en estas funciones son enteros de 8 bits sin signo. Para enviar la información de los datos provenientes del sistema de visión se crea una clase denominada *ElementosAlt*, parte del código es el siguiente:

```
%% Eliminación de Objetos Actuales en RobotStudio
pos=[4 1 1 1 1 1 1 1];
robot.ElementosAlt(pos);

%% Creación y Posicionamiento de Cilindros Rojos en RobotStudio
  for n=1:circlesR
    pos=[4 1 double(eval(['CircleObjR' num2str(n)]))];
    robot.ElementosAlt(pos);
  end
%% Creación y Posicionamiento de Cilindros Verdes en RobotStudio
  for n=1:circlesG
    pos=[4 2 double(eval(['CircleObjG' num2str(n)]))];
    robot.ElementosAlt(pos);
  end ….
```

Finalmente, se crea la clase *MovimientoAltCI*, la cual contiene la información correspondiente a la posición que deberá tener cada objeto en función de su forma y color, el siguiente es parte del código de la clase:

```
%% Movimiento de Cilindros Rojos en RobotStudio
alt=0;
  for n=1:circlesR
    pos=double(eval(['CircleObjR' num2str(n)]));
    a=double(eval(['CircleObjR' num2str(n)]));
    a(6)=pos(6)+10;
    c=pos(6)+alt;
    robot.CI(a);
    pause(1)
    robot.Ventosa(0)
    robot.CI(pos);
    pause(1)
    robot.Ventosa(1)
    robot.CI(a);
    pause(1)
    robot.CI([180 0 180 300 1 350]);
    pause(1)
    robot.CI([180 0 180 150 -200 10+c]);
    pause(1)
    robot.CI([180 0 180 150 -200 c]);
    pause(1)
    robot.Ventosa(0)
    robot.CI([180 0 180 150 -200 10+c]);
    pause(1)
    robot.CI([180 0 180 300 1 350]);
    pause(1)
    alt=alt+pos(6)+1;
  end
```

ISSN-2531-2960

Resultados experimentales

En esta sección, se analiza cómo puede utilizarse la plataforma para abordar el tema de la cinemática inversa de manera experimental a través del uso de la plataforma utilizando la aplicación de tomar y colocar objetos ordenándolos por color y forma en una posición deseada previamente programada en el código RAPID en RobotStudio. Inicialmente, el sistema de visión captura la información del entorno real donde se encuentran los objetos, obteniendo su posición y orientación como se explicó en la sección anterior. Posteriormente, la información del sistema de visión se utiliza únicamente para crear estos objetos en el entorno de simulación a través de ejecutar el programa *ElementosAlt*, ver figura 6.

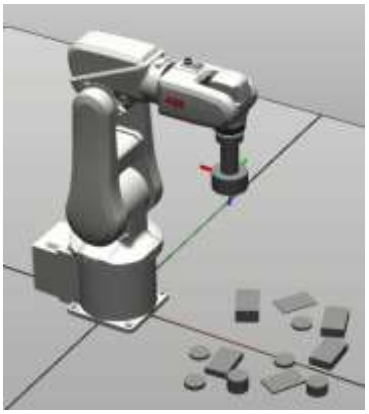

Figura 6 Objetos creados en RobotStudio

Fuente: *Elaboración propia*

Para finalizar, se ejecuta el programa de *MovimientoAltCI*, en el cual se ha implementado el cálculo de la cinemática inversa del robot que permite obtener el valor de cada articulación necesario para alcanzar los objetos y colocarlos en la posición deseada. En las figura 7 a la 10 se muestra el movimiento de los objetos que tiene un orden predeterminado, primeramente, se posicionan los cilindros rojos, después los verdes y por último los azules; una vez terminado, se comienza con los tetraedros, comenzando por el rojo, después el verde y para finalizar el azul.

Las imágenes muestran el caso donde la altura es variable por tal motivo se pierde el color de los objetos en el simulador.

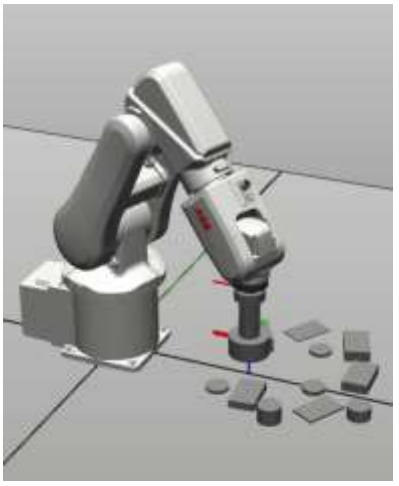

Figura 7 Inicia con cilindros rojos

Fuente: *Elaboración propia*

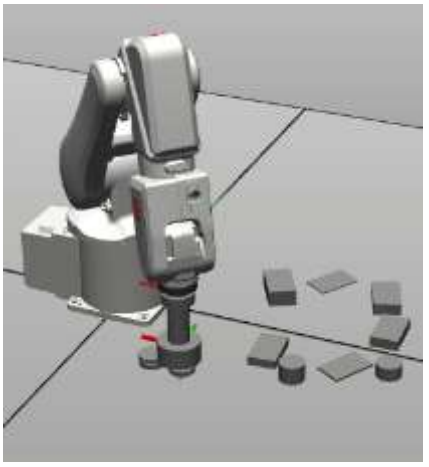

Figura 8 Continúa con cilindros verdes

Fuente: *Elaboración propia*

ISSN-2531-2960

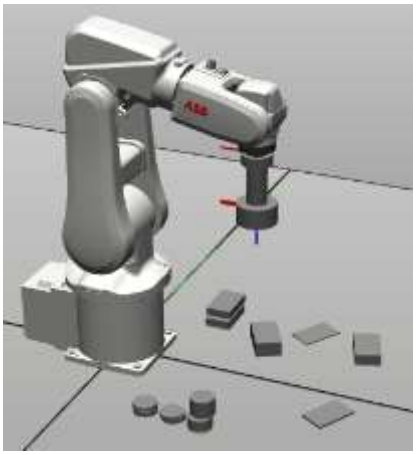

Figura 9 Continúa con Tetraedros rojos

Fuente: *Elaboración propia*

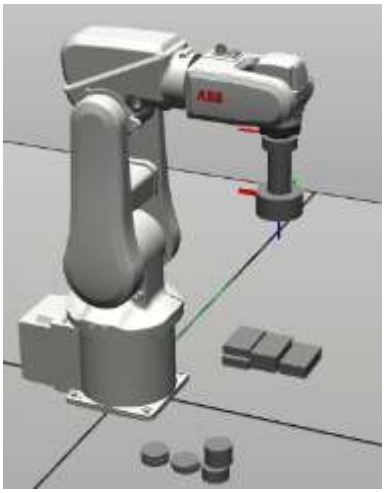

Figura 10 Finalmente Tetraedros verdes y azul

Fuente: *Elaboración propia*

Para comprobar experimentalmente que los valores articulares de cada articulación obtenidos de la cinemática inversa son correctos se realiza su monitoreo a través del ambiente de Simulink graficándolos junto con la información de la posición articular real que se obtiene de RobotStudio, en la figura 11, se muestra únicamente la gráfica correspondiente a la primera articulación del robot. La línea azul punteada corresponde a la posición articular teórica y la línea negra continua corresponde a la posición real de cada articulación. Inicialmente, el robot parte de la posición de home, posteriormente se mueve a la posición donde está el objeto para finalmente moverse a la posición deseada pasando por la posición de home.

Como puede observarse la posición articular calculada teóricamente (línea azul punteada) y la posición real del robot (línea negra continua) coincide, lo cual permite concluir que el análisis teórico realizado es correcto, la diferencias mostradas en la curvas durante los transitorios es debida al tiempo de muestreo entre Simulink y RobotStudio.

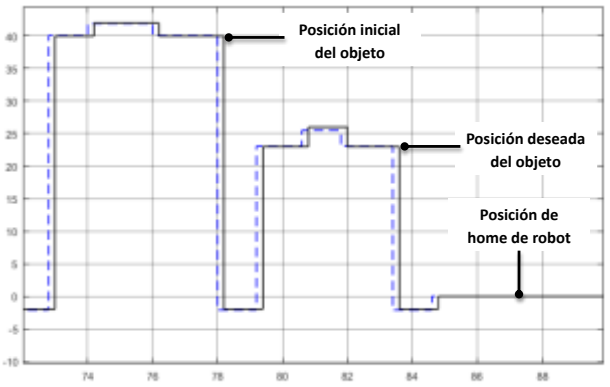

Figura 11 Movimiento de la primera articulación

Fuente: *Elaboración propia*

Agradecimiento

Los autores desean agradecer al Tecnológico Nacional de México por el financiamiento otorgado para la realización del proyecto. Así mismo, agradecer al Instituto Tecnológico Superior de Uruapan y a la Universidad Politécnica de Uruapan por las facilidades otorgadas para la realización del mismo.

Conclusiones

En este trabajo se propone un Laboratorio virtual de robótica basado en los programas de Matlab y RobotStudio su potencial como módulo didáctico es evaluado a través de la aplicación de organización de objetos por color y forma que pueden ser detectados en el área de trabajo de un robot industrial a través de un sensor Kinect. Este módulo didáctico permite desarrollar distintas capacidades sobre temas como visión artificial, programación y control de robots articulados, comunicación por distintos protocolos, modelado matemático de brazos robóticos, entre otros temas.

ISSN-2531-2960

La comunicación desde Matlab está planeada para ser compatible con el controlador de cualquier robot ABB, ya sea físico o simulado. La implementación de este módulo permite la creación de distintas aplicaciones, y genera la posibilidad de crear distintos módulos virtuales, lo que llevaría a la implementación de un laboratorio virtual más complejo, que incluya una gran gama de aplicaciones. Esto nos permite concluir que con la ayuda de la plataforma de simulación se refuerza la comprensión de los conceptos fundamentales en robótica ya que resulta ser un mecanismo de realimentación para que los estudiantes detecten las posibles desviaciones hechas en sus análisis teóricos cuando los implementan en la plataforma propuesta.

Referencias

Žlajpah, L. (2008). Simulation in robotics. Mathematics and Computers in Simulation, 79(4), 879-897.

Connolly, C. (2009). Technology and applications of ABB RobotStudio. Industrial Robot: An International Journal, 36(6), 540-545.

Vila-Rosado, D. N., & Dominguez-Lopez, J. A. (2005). A matlab toolbox for robotic manipulators. In Computer Science, 2005. ENC 2005. IEEE Sixth Mexican International Conference on, 256-263.

Gutiérrez Corbacho, A., (2014). Trabajo de Fin de Grado: "Desarrollo de una interfaz para el control del robot IRB120 desde MATLAB", Alcalá de Henares: Universidad de Alcalá.

Frydrysiak M., Trabajo de Fin de Grado: "Socket based communication in RobotStudio for controlling ABB-IRB120 robot. Design and development of a palletizing station", Alcalá de Henares: Universidad de Alcalá.

Lazaro-Arvizu, Y., Morales-Caporal, R., Ordoñez-Flores, R., Quintero-Flores, P. Y., Leal-López, M., (2015). Desarrollo de un software para la simulación y control de un robot industrial. Revista de Tecnología e Innovación, 2-5, 958-967.

Corke P., (1996). A robotics toolbox for Matlab, IEEE Robotics and Automation Magazine 3, 24–32.

Gil Arturo, Oscar Reinoso, José Maria Marin, Luis Paya and Javier Ruiz, (2015). Development and deployment of a new robotics toolbox for education. Computer Applications in Engineering Education, 23(3), 443-454.

González-Palacios, M. A., González-Barbosa E. A. and L. A. Aguilera-Cortés, (2013). SnAM: A simulation software on serial manipulators, Engineering with Computers Springer 29, 87–94.

López-Nicolás, G.; Romeo, A.; Guerrero, J. J., (2009). Project based learning of robot control and programming. En International Conference on Engineering Education and Research, Korea.

Damić, Vjekoslav; Carića, Ćira; Husić, Maida Ĉohodar, (2014). Procedure for Visualization and Dynamic Analysis of Robot Manipulator, Journal of Trends in the Development of Mahinery and Associated Tecnhnology, 18(1), 127-130.

ISSN-2531-2960