REVISTA PERSPECTIVAS

VOLUMEN 3, N°1 / ENERO - JUNIO 2021 e-ISSN: 2661-6688

Interfaz gemelo digital de un brazo robótico UR3 para repasar el marcado de piezas para corte de superficies

Digital twin interface of an UR3 robotic arm to go over the pieces marking for cutting of surfaces

[1,&]Alexis Mariño-Salguero, [1,#]Sofía Sánchez-Urresta, [2,♫]Hector Espinoza-Velasco, [3,☼]Santiago Martínez

1Facultad de Informática y Electrónica, Escuela Superior Politécnica de Chimborazo, Riobamba, Ecuador
2Facultad de Ingeniería, Universidad Nacional de Chimborazo, Riobamba, Ecuador
3Departamento de Ingeniería Mecánica, Universidad de los Andes, Bogotá, Colombia

[&]alexis.marinio@espoch.edu.ec, [#]sofia.sanchez@espoch.edu.ec, [♫]hector.espinoza@unach.edu.ec, [☼]s.martinezc@uniandes.edu.co

Resumen- **Hoy en día la industria manufacturera tiene que hacer frente a cambios paradigmáticos de producción, donde la personalización en masa son pilares fundamentales para adaptarse a la demanda cambiante del mercado. La tecnología actual brinda las herramientas necesarias que, al integrarse con las habilidades humanas, da paso a las *Smart Industry* entrando así a un entorno de revolución industrial 4.0. En el presente trabajo se propone el diseño de una interfaz *Digital Twin* que apoya la actividad de supervisión del brazo robótico UR3 mientras este realiza el marcado para corte de superficies en el contexto de producción flexible. La interfaz se logra a través de una interacción Humano-Computador-Máquina (HCMI) mientras que la autoadaptación a la producción y cambios ambientales se consigue mediante un algoritmo *cut and packing* y una cámara adaptada al robot respectivamente. La interfaz se implementa y valida mediante un estudio de caso de laboratorio virtual en el que se observa que el Digital Twin replica el movimiento del robot con un tiempo de latencia insignificante.**

Palabras Clave- **Cambios Paradigmáticos, Smart Industry, Digital Twin, Producción Flexible, Cut and Packing Algorithm.**

Abstract- **Nowadays manufacturing industry is facing paradigm production changes, where mass customization is a fundamental pillar in adapting to changing market demands. Today's technology provides the necessary tools that, when integrated with human skills, give way to the Smart Industry thus entering an environment of industrial revolution 4.0. This paper proposes the design of a Digital Twin interface that supports the monitoring activity of the UR3 robotic arm while it performs the marking for surface cutting in the context of flexible production. The interface is achieved through a Human-Computer-Machine interaction (HCMI), while the self-adaptation to production and environmental changes is achieved through a cut and packing algorithm and a camera adapted to the robot respectively. The interface is implemented and validated through a virtual laboratory case study in which it is observed that the Digital Twin replicates the movement of the robot with a negligible latency time.**

Keywords- **Paradigmatic Changes, Smart Industry, Digital Twin, Flexible Production, Cut and Packing Algorithm.**

I. INTRODUCCIÓN

La industria manufacturera en la actualidad enfoca su desarrollo hacia la cuarta etapa de industrialización. Esta etapa conocida como *Industria 4.0* se define como una combinación de tecnologías y enfoques metodológicos para la resolución de desafíos actuales y proyectados hacia el futuro [1]. El paradigma de la Industria 4.0 está formado por tres factores mutuamente interconectados: los sistemas ciberfísicos, el Internet de las cosas y la nube [2]. Al tener acceso a estos elementos fácilmente se puede crear ambientes virtuales que son espacios tridimensionales, reales o imaginarios, generados por computadora, con los que el usuario puede interactuar [3]. La principal característica busca que esta interacción sea lo más natural posible, por esto se usa medios electrónicos que permiten al usuario la manipulación del ambiente. Los dispositivos pueden brindar una experiencia visual (no inmersiva) o multisensorial (inmersiva) de una simulación computarizada en tiempo casi real. Las interfaces y simulaciones estrechamente vinculadas con el espacio de trabajo son un claro ejemplo de un ambiente virtual no inmersivo [4]. Los beneficios al utilizar este tipo de ambientes son: la libertad y amplitud de movimiento en la escena virtualmente generada, la visualización, retroalimentación táctil, los detalles y escalabilidad. En el caso de la manufactura, el poder analizar los objetos en escala real, permite tomar decisiones, realizar y observar las modificaciones en el espacio del objeto [5].

Estas innovaciones de fabricación y servicio basadas en ambientes inmersivos y no inmersivos [6] son tendencias y desafíos inevitables para las industrias manufactureras ya que, hoy en día, las cambiantes demandas del mercado, los lotes pequeños de productos personalizados producidos en masa con requerimientos de entrega rápida se consideran uno de los principales desafíos que se presentan ante sus sistemas de producción y fabricación [7].

Este cambio requiere la utilización de herramientas informáticas predictivas inteligentes, de modo que los datos puedan procesarse sistemáticamente en información para explicar las incertidumbres y, por lo tanto, tomar decisiones

 Fecha de Recepción: 02/Dic/2020. Fecha de Aceptación: 17/Dic/2020. DOI: 10.47187/perspectivas.vol3iss1.pp32-42.2021

más informadas y acertadas al momento de la producción [8]. Ya que, los futuros sistemas de fabricación necesitarán ser más autónomos estas herramientas inteligentes deberán ejecutar tareas de alto nivel sin control humano, capaces de decidir entre un conjunto de alternativas para tomar decisiones y ejecutar habilidades. Para que esto suceda los sistemas autónomos necesitan tener acceso a modelos realistas del estado actual del proceso y su propio comportamiento en relación con su entorno y el mundo real así llamado Digital Twin [9].

En una fábrica que pertenece a la Industria 4.0, las máquinas están conectadas como una comunidad colaborativa, por lo que los resultados de los procesos de producción se pueden analizar manualmente o con ayuda de simulaciones. Por lo tanto, para su mejora se requiere la investigación, la adquisición de datos digitales y automatizados y del concepto de Digital Twin [10]. Así, el proceso de producción permitirá el acoplamiento del sistema de producción con su equivalente digital como base para una optimización con un retraso reducido al mínimo entre el momento de la adquisición de datos y la creación de la Digital Twin[11]. Como consecuencia un sistema de producción-cibernética física puede ser generada y así asegurar una concordancia máxima del proceso ciberfísico con su modelo de la vida real convirtiendo a un Digital Twin en un componente básico en la industria 4.0 [12].

Lo restante del presente documento se organiza como sigue. En la sección II se presenta el proceso metodológico de desarrollo del presente trabajo para diseñar una interfaz Digital Twin de un brazo robótico UR3 en el contexto de producción flexible en un entorno cambiante. En la sección III se describen los resultados obtenidos en la investigación comenzando por el algoritmo de corte implementado, se detallan también los resultados del giro y adaptación al ambiente de trabajo para comprobar la fidelidad de las coordenadas enviadas al robot para su puesta en funcionamiento. Por último, se obtiene datos de tiempo de latencia para calcular el retardo existente entre el *Digital Twin* y el robot.

II. METODOLOGÍA

Previo al proceso de autoadaptación de producción y ambiente, se considera una línea de producción en la cual está instalado el brazo robótico UR3, el mismo que está encargado de realizar cortes según los requerimientos de producción a planchas que arriban a su espacio de trabajo en una posición y orientación arbitraria. En la Fig.1 se muestra un modelado CAD del robot en su ambiente de trabajo.

El Digital Twin propuesto debe ser una réplica exacta del robot y adaptarse a un entorno de trabajo cambiante. Para lo último se utiliza una adaptación del robot llamada Wrist Camera que junto al software de programación "Camera Locate" le ofrece a Polyscope (software de programación del brazo robótico UR3) una interfaz de enseñanza de objetos que es capaz de reconocer la posición y ubicación de estos en base al sistema de referencia de la base del robot.

La orden de producción son piezas rectangulares requeridas por el operador las cuales mediante un algoritmo de Cut and Packing buscan la mejor distribución siempre priorizando que exista el menor desperdicio posible. Posteriormente, dicha distribución será llevada a la configuración de la plancha real y dará paso a la réplica virtual.

La metodología usada es *Methodology for Multi-Robot Manufacturing Cell Commissioning*, esta se basa en cuatro fases con el fin de determinar un proceso de virtualización que puede ser migrado a un proceso real. Esta metodología servirá de referencia para el diseño de la interfaz Digital Twin y sus fases son: Diseño, Modelo VR, Implementación y Digital Twin.

A. Diseño

Se propone una correlación optima entre el operador, el *Digital Twin*, la planta de producción y el entorno de trabajo, es decir, se tiene una interacción humano-computadora- máquina (HCMI) que puede aplicarse a cualquier entorno de fabricación donde pequeños lotes de productos personalizados son una de sus características. La Fig.2 muestra la arquitectura HCMI propuesta para ayudar al operador en la "actividad de supervisión".

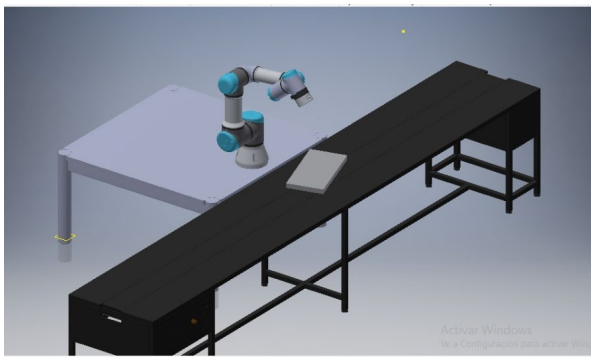


Fig. 1. Robot UR3 y su ambiente de trabajo propuesto.

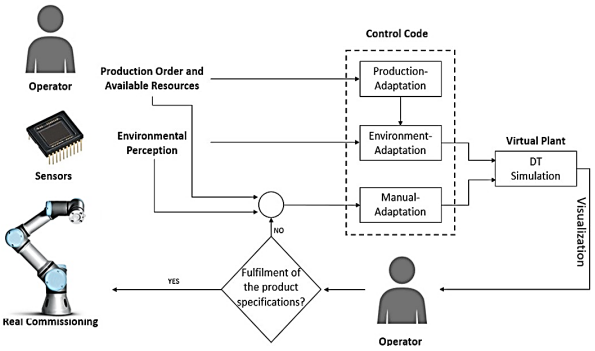


Fig. 2. Arquitectura para la interfaz *Digital Twin* en el proceso de corte de piezas

El proceso da inicio con los requerimientos de corte que son ingresados por el operador, estos especifican los recursos disponibles y la orden de producción que da paso a la operación de fabricación. Con la orden de producción definida, la percepción y adaptación del entorno es realizada mediante la *Wrist Camera*. Los datos generados por el operador y la cámara son enviados hacia el controlador que mediante bucles de ajuste auto-adapta la lógica de su control para cumplir la orden de producción con los recursos a disposición. Terminada esta acción, el *Digital Twin* inicia una réplica de las configuraciones de la actividad de fabricación. Dicha acción es supervisada por un operador que valida la

factibilidad de la operación (por ejemplo, movimientos bruscos, colisiones, adaptación inadecuada al entorno, etc.).

La supervisión visual del operador llevará a dos posibles resultados. En el primer resultado, la operación de fabricación se puede realizar con éxito y los productos satisfacen las especificaciones de la orden de producción. En este caso, el supervisor da paso al *Real Commissioning* y el proceso termina. En el segundo resultado, existe una falla en la operación de autoadaptación debido a que la orden de producción no es factible o el producto no cumple con las especificaciones requeridas. Esto crea la necesidad de adaptar la lógica de control de manera manual. A continuación, los datos generados (el movimiento del *Digital Twin,* el código lógico de control, la orden de producción, los recursos disponibles y la percepción del entorno) serán utilizados como base para que un ingeniero en software pueda diseñar una nueva lógica de control que sea viable y comenzar un nuevo ciclo de validación. Sin embargo, la adaptación manual se plantea como trabajo futuro.

B. Modelo VR

1. Algoritmo de Cut and Packing

Para la elección del algoritmo de corte se tiene en consideración varios factores:

- Dimensionalidad
- Forma de piezas
- Orientación de piezas •

Función objetivo

Para el caso de estudio se considera un problema bidireccional con piezas regulares las cuales tiene la posibilidad de ser rotadas 90° y una función objetivo que busca maximizar la salida. La tabla 1 muestra las diferentes características de los algoritmos considerados para la resolución de este problema.

TABLA.1. ALGORITMOS DE CORTE VS FACTORES DETERMINANTES.

Algoritmo	Maximización Salida	Minimización Entrada	Piezas Rectangulares	Generación de Columnas	Patrón de corte Libre	Fácil Implementación
Algoritmo Grasp		X	X		X	
Algoritmo Gloton		X	X	X		X
Algoritmo Viswanathan y Bagchi		X			X	
Algoritmo Genético	X			X	X	
Algoritmo Corte Guillotina	X		X	X		X
Heurísticos	X				X	
Branch And Bound	X			X	X	X

A la luz de la comparación de dichos factores se decantó por el *Algoritmo Corte Guillotina* que presenta las mejores características dentro del tema de estudio y en comparación con sus competidores. El mismo que se implementó dentro del software Python y funciona de acuerdo con el diagrama de flujo mostrado en la Fig.3.

Los datos de ingreso hacia el algoritmo son: los recursos disponibles y la orden de producción. En este caso, el recurso disponible es una plancha rectangular de ancho W y largo L mientras que la orden de producción es definida por las piezas que requiere el operador de dimensiones [wj, lj]. A continuación, se hace una preclasificación que enlista las piezas de manera descendente según su área y pasa a una condición en donde el algoritmo posiciona todas las piezas una por una o ya no existe espacio en la plancha, esto se realiza según la heurística que en esta aplicación es *best long side fit.* Entonces el algoritmo termina su ejecución generando como datos de salida las coordenadas cartesianas de las 4 esquinas de cada una de las piezas rectangulares finales. Los puntos están definidos con referencia al punto (0,0) ubicado en la esquina inferior izquierda de la plancha original como se muestra en la Fig.4.

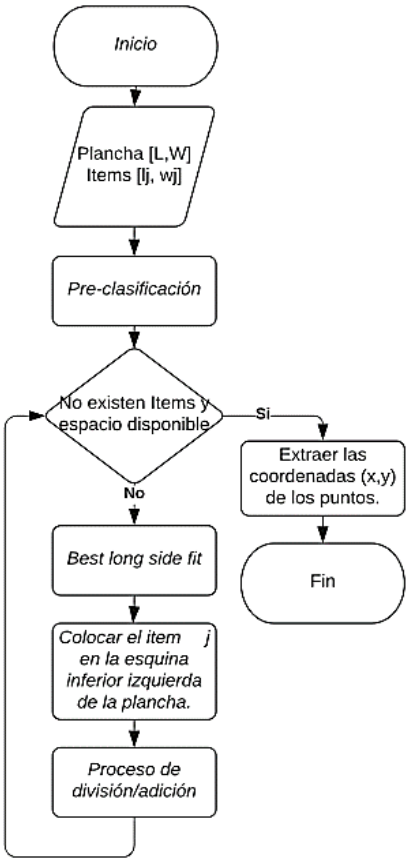


Fig.3. Diagrama de flujo del algoritmo de corte

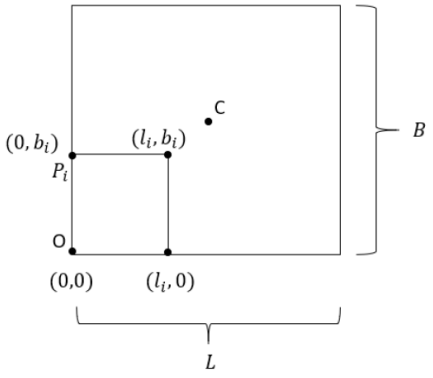


Fig.4. Ejemplo de los puntos cartesianos generados por el algoritmo

2. Percepción del entorno de trabajo

La ubicación y orientación inicial de las planchas que llegan al espacio de trabajo del robot son captadas con la adaptación hardware llamada Wrist Camera que es almacenada en una variable definida como *object_location* la cual posee 6 datos.

$$object\_location = [X_0, Y_0, Z_0, x_{rotación}, y_{rotación}, z_{rotación}]$$

Donde los tres primeros datos son la posición del centro del objeto con relación al sistema de referencia de la base del robot y los 3 siguientes son la orientación con relación al sistema de referencia de la base del robot donde los ejes *x* rotación e *y* rotación son paralelos al plano de trabajo en el que se realizó la calibración y el eje *z* es normal al plano formado por los mismos y apunta hacia abajo Fig.5(a).

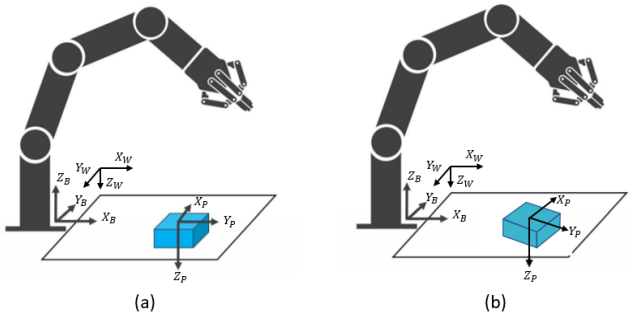


Fig.5.(a) Sistemas de referencia del robot y la pieza sin rotar (b) Sistemas de referencia del robot y la pieza rotada.

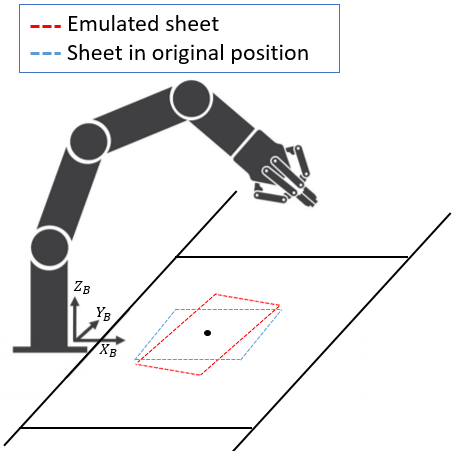


Fig.6. Robot UR3 en posición para adquirir los datos mediante la cámara.

El plano formado por los ejes Y*w* e X*w* siempre será paralelo al plano formado por los ejes Y*p* e X*p* sin importar la orientación en la que llegue la plancha por lo que la rotación de esta se da sobre el eje Z*p* como se muestra en la Fig.5 (b). El dato brindado por la cámara nos muestra esta rotación menos 90 grados que representan la rotación entre el sistema de referencia del plano de trabajo y el de la plancha. Los sistemas de referencia y la pieza rotada se muestran en la Fig.5.

Los datos que entrega el algoritmo de corte no tienen ninguna relación con el robot o la plancha. Lo que se busca con la percepción del entorno de trabajo es que el algoritmo funcione en un entorno cambiante y no solo como un proceso computacional ideal.

Para trasladar los puntos obtenidos por el algoritmo de corte hacia la plancha real, se asume una plancha de tamaño (L x W) perpendicular al sistema de referencia de la base del robot y ubicando su centro en la coordenada (x_0,y_0,z_0) obtenida por la cámara como muestra la Fig.6.

A todos los puntos obtenidos por el algoritmo se les suma la coordenada de la esquina inferior izquierda de la plancha asumida (X_p,Y_p,Z_p) que se muestra en la Fig.7, la cual se consigue con la ecuación (1).

$$(X_p\ Y_p\ Z_p) = \left(-\frac{W}{2}, -\frac{L}{2}, Z_0\right) \quad (1)$$

De esta forma todos los puntos quedarán con respecto al sistema de referencia de la base del robot.

Al ubicar los puntos en la plancha emulada quedan paralelos al sistema de referencia de la base del robot. A continuación, se debe rotar estos para llegar a la posición real.

Para realizar la rotación se toma como eje el centro de la plancha, por ende, todos los puntos deben estar referenciados con respecto al mismo, esto se lo consigue médiate una resta de vectores siguiendo la ecuación (2) a través de todos los puntos, como se muestra en la Fig.8.

$$\overrightarrow{V_{CP}} = \overrightarrow{V_{BP}} - \overrightarrow{V_{BC}} \quad (2)$$

$\overrightarrow{V_{CP}}$ = Vector desde el centro de la pieza al punto P.

$\overrightarrow{V_{BP}}$ = Vector desde la base del robot hasta el punto

$\overrightarrow{V_{BC}}$ = pVector desde la base del robot hasta el centro de la pieza (dato obtenido por la cámara).

Con los puntos referenciados al centro de la plancha se aplica la matriz de rotación mostrada en la ecuación (3), como se observa en la Fig.9.

$$\begin{bmatrix} X_R \\ Y_R \\ Z_R \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} x \begin{bmatrix} X_n \\ Y_n \\ Z_n \end{bmatrix} \quad (3)$$

Por último, para poder utilizar los puntos previamente rotados tienen que ser nuevamente referenciados a la base del robot lo que se consigue utilizando una vez más la ecuación (2) despejando esta vez $\overrightarrow{V_{BC}}$.

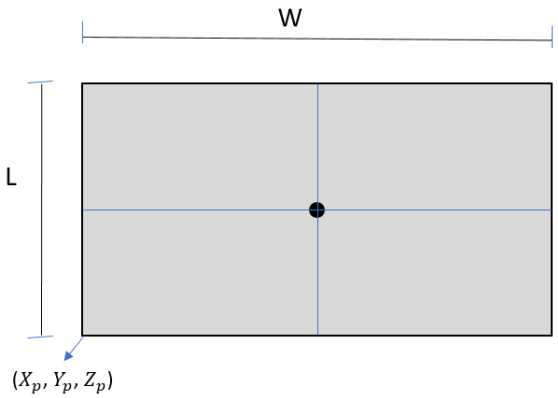


Fig.7. Representación de la plancha asumida.

REVISTA PERSPECTIVAS
e-ISSN: 2661-6688

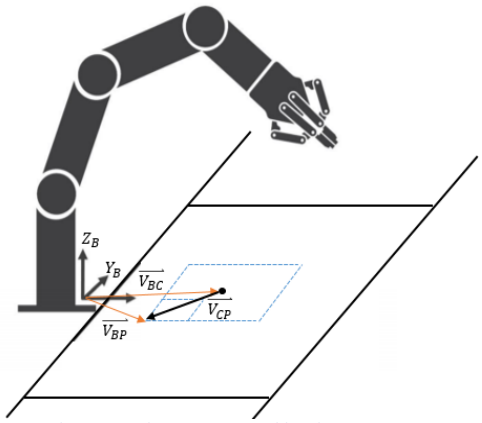


Fig.8. Resta de vectores para ubicar los puntos.

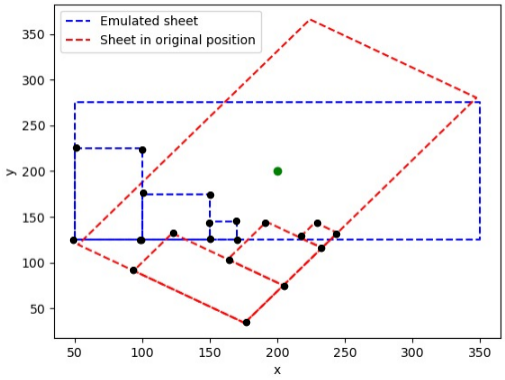


Fig.9. Puntos generados por el algoritmo y ubicados en la plancha real.

3. *Virtualización del brazo robótico UR3 y entorno*

Para la tarea de virtualización se utilizó el software *Experior*. La empresa desarrolladora del software es un partner comercial de la compañía *Universal Robots*, por lo que posee un catálogo que contiene sus productos como son los brazos robóticos UR3, UR5 y UR10 los mismos que pueden ser configurados para trabajar como *Digital Twin*. La Fig.10 muestra el brazo robótico UR3 en la interfaz de *Experior*.

Este modelo 3D es una réplica exacta tanto física como funcional del robot UR3. La comunicación entre el robot físico y virtual se da mediante protocolo Modbus TCP/IP donde los dos deben poseer la misma dirección y la posición del robot es configurada de acuerdo con la ubicación real del robot.

El efector final "Gripper" como de la mesa fueron modelados en el software AutoCAD donde se asigna el material de construcción, aluminio, y sus dimensiones las cuales son a escala real. Los archivos .dxf, que genera AutoCAD, son exportados de forma directa hacia *Experior* como se observa en la Fig.11. En este punto, se requiere configurar un nuevo catálogo que contenga este nuevo modelo, lo que se consigue ingresando a My Catalog y accediendo a la opción Add New Tool.

En el caso de la mesa de trabajo se asignarán sus propiedades mediante código C# subyacente a *Experior*, donde, al ser un objeto estático inanimado la única configuración asignada es la posición.

El Gripper está relacionado al funcionamiento del robot por lo que es necesario asignar comportamiento al mismo. Una de las funciones necesarias en su programación es *Grabbing Properties* que permite un acople entre objetos. En este caso, es necesario aumentar un módulo de ubicación y orientación que contiene la posición *z*, *x*, e *y* en [mm] como los ángulos *yaw*, *pitch* y *roll* en grados que junto a la función mencionada trasmitirán el movimiento del efector final del robot hacia el Gripper. Para esto solo es necesario unir el robot con el Gripper y el acople se da de forma automática al igual que su movimiento.

Por último, al tratarse de un entorno cambiante la plancha es diseñada y programada en C# ya que sus dimensiones se modifican según los recursos disponibles (ver Fig.12).

La plancha, como se denotó en párrafos anteriores, está ubicada y orientada con respecto al sistema de referencia del robot y localizada sobre el plano de trabajo (mesa). Por tanto, se asigna una posición alterna que contenga la ubicación del robot dentro de *Experior* para que la plancha este referenciada con respecto a la base del robot.

La plancha en *Experior* tiene que ser simulada digitalmente y representar la realidad. Para esto, es necesario adquirir datos los cuales son obtenidos por la cámara y enviados a *Experior* mediante protocolo Modbus creado dentro de PolyScope, lo que se explicara más adelante en la fase de implementación. Dentro de *Experior* se configura un módulo de entrada que contenga el ángulo, la posición *x* e *y* del centro de la plancha y dos variables booleanas que serán de utilidad para la programación.

Además, en *Experior* es necesario crear una comunicación Modbus, con la dirección IP del robot, con este protocolo se conecta el módulo de entradas con los datos enviados de PolyScope. A cada entrada se asigna una dirección y el nombre de la variable a la cual se conectará.

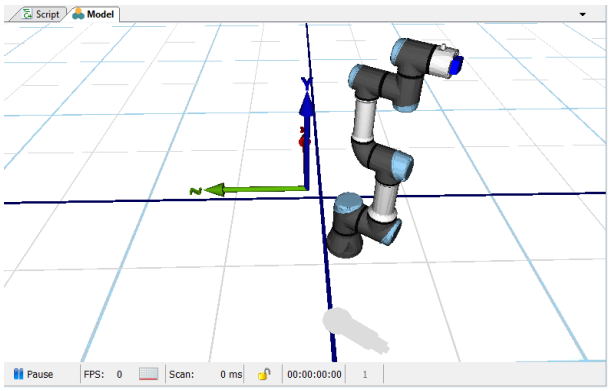


Fig.10. Brazo robótico UR3 dentro del software *Experior*

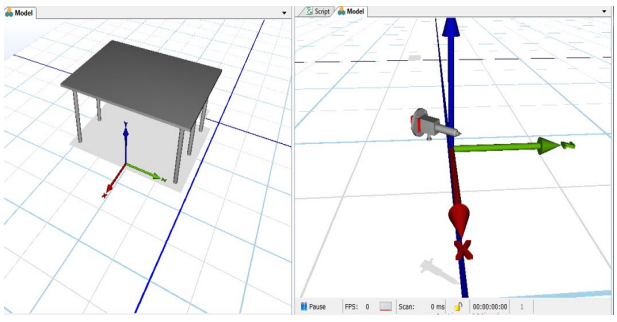


Fig.11. Mesa de trabajo y Gripper para marcado de piezas.

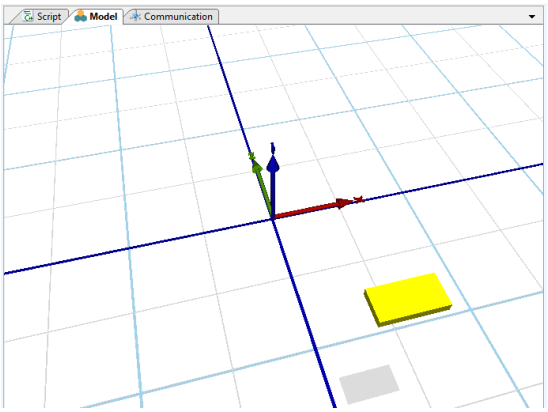


Fig.12. Diseño de la plancha en Experior

C. Implementación

1. Manejo de la cámara y el software Camera Locate

El software Camera Locate ofrece una interfaz muy intuitiva para poder configurar el sistema de visión artificial. A este software se accede por la interfaz principal de PolyScope.

La función que se configura es la enseñanza de objetos paramétricos. Este método cosiste en un modelo basado en parámetros de una forma 2D básica (círculo, anillo, cuadrado y rectángulo). Lo que se busca es que el sistema reconozca y ubique con gran precisión objetos que tienen características semejantes. Una vez que se define la posición de la fotografia el operador puede utilizar el nodo de la localización de la cámara dentro de un programa al igual que la variable *object_location* que contiene la información de ubicación y rotación.

En el caso de estudio que se reporta se configura el sistema para que reconozca la plancha rectangular de dimensiones 300x150 [mm]. A continuación, se ubica la plancha en diferentes posiciones donde la cámara toma capturas y mediante un algoritmo de detección de bordes el software aprende a localizar la ubicación y rotación de la plancha. Luego se crea el nodo Camera Locate dentro del entorno de programación de PolyScope lo que permite trabajar con estos datos.

2. Código en PolyScope

El programa realizado en PolyScope es utilizado para la adquisición de datos como: requerimiento de corte, recursos disponibles y la percepción del entorno. El diagrama de flujo mostrado en la Fig.13 describe la secuencia lógica para realizar dicho proceso.

El programa comienza ubicando al robot en la posición establecida para que la cámara pueda captar el entono.

Para trabajar con los datos, estos deben ser enviados desde el robot hacia un ordenador para ello se utiliza una comunicación cliente-servidor TCP/IP usando el brazo robótico UR3 como cliente. Inicialmente se abre una conexión entre el robot y la computadora llamada "Socket", dentro del cual se agrega la dirección IP del robot, un puerto arbitrario que no esté siendo usado por el otro dispositivo y el nombre del programa que administrará el socket en el otro dispositivo al cual se va a conectar el robot.

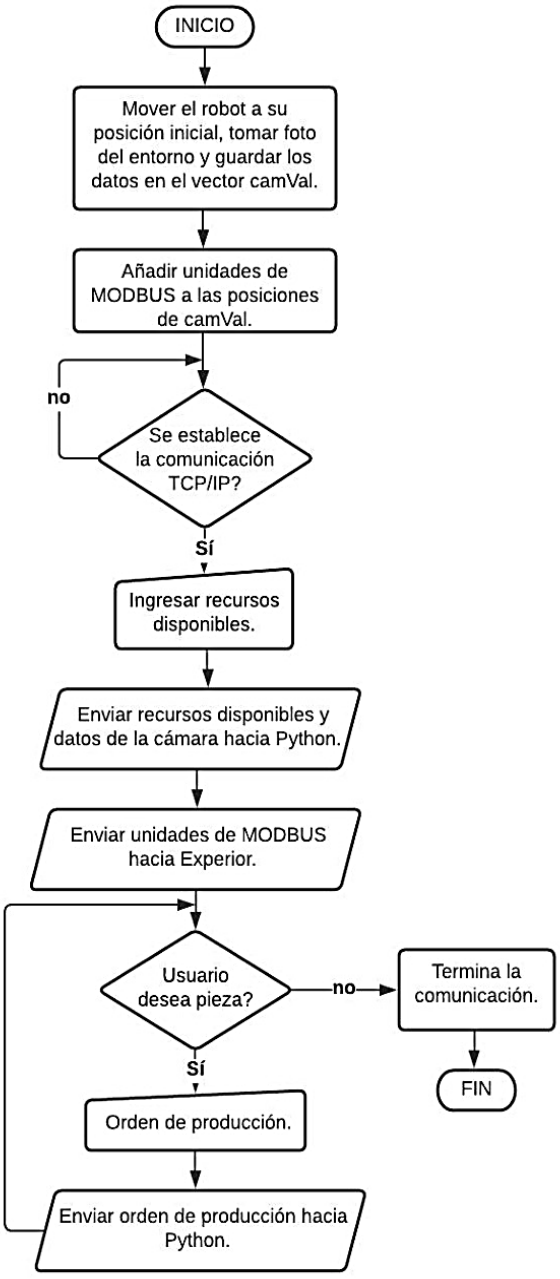


Fig.13. Diagrama de flujo del algoritmo en PolyScope

El entorno dentro *Experior* necesita los datos obtenidos por la cámara para que la plancha simulada represente fielmente la ubicación y orientación de la real. Los datos tienen que enviarse desde el robot hacia *Experior* lo cual se consigue usando protocolo Modbus. Esta configuración se encuentra dentro del menú *Installation – Fieldbus – Modbus* dentro de PolyScope. Para esta configuración, se asigna la misma dirección IP del robot, se crean tres *Register Output* para enviar datos numéricos que contengan las posiciones en los ejes *X* e *Y*, y el ángulo en el que se encuentra orientada la plancha. Los datos deben ser tratados antes de ser enviados ya que *Experior* solo trabaja con datos enteros y positivos para lo cual se crean dos *Digital Output* que indiquen cuando las posiciones de la plancha sean negativas. A cada unidad de Modbus creada se le asigna una dirección arbitraria como se muestra en la Fig.14.

REVISTA PERSPECTIVAS
e-ISSN: 2661-6688

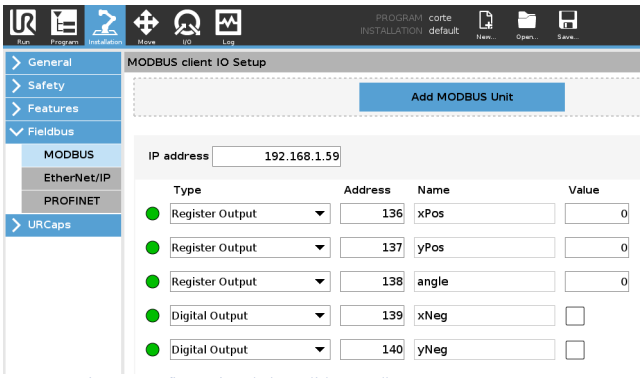


Fig.14. Configuración de las salidas Modbus para *Experior*.

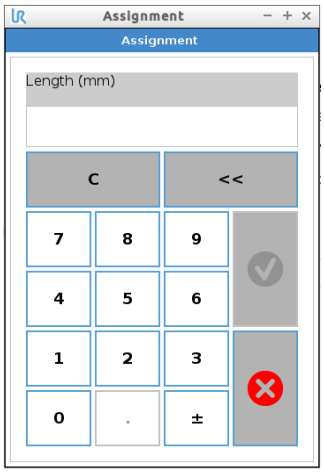


Fig.15. Cuadro de entrada de datos.

La entrada de los recursos disponibles y los requerimientos de corte se dan mediante *Assignment* configurados como *Operator*. Esto nos permite crear cuadros de entrada de datos y guardarlos como variables para luego ser utilizadas como se muestra en la Fig.15.

Para enviar los datos hacia Python se usa la función *Socket_sent_string*(), la cual envía los datos como una cadena de caracteres. Lo que se pretende es crear un vector que incluya los datos de la cámara, el tamaño de la plancha y todos los requerimientos de corte para que el algoritmo de control desarrollado en Python devuelva como resultado el movimiento que tiene que realizar el robot. Dicho algoritmo se detalla en la siguiente sección.

Finalmente, se usa la función *socket_close( )* que termina la comunicación entre el brazo robótico y la computadora dando paso al algoritmo de corte desarrollado en Python que posteriormente controlará el robot enviando comandos de movimiento a PolyScope.

3. Desarrollo del algoritmo de control en Python.

El programa realizado en Python complementa el proceso de marcado para corte de superficies, el mismo que es utilizado para la recepción de datos generados en PolyScope, tratar los datos mediante el algoritmo de corte, manipular los resultados del algoritmo para que se ajusten al ambiente de trabajo real y, finalmente, poner en funcionamiento al robot real. El diagrama de flujo mostrado en la Fig.16 describe la secuencia lógica para realizar el proceso de marcado de piezas.

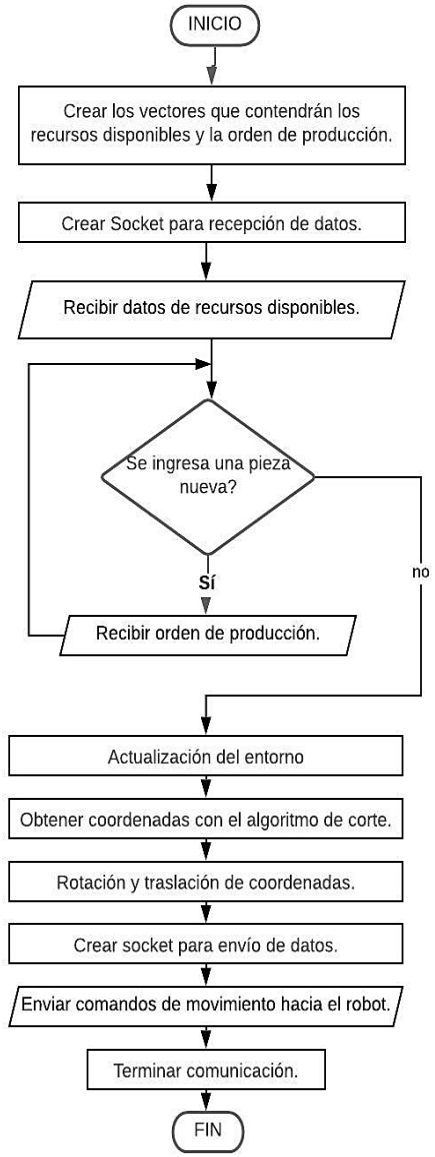


Fig.16. Diagrama de flujo del algoritmo en Python desarrollado para el marcado de piezas

Paso 1: El programa da inicio creando 4 vectores:

- datT = [] Lista con los datos del tablero
- datP = [] Lista con los datos de las piezas
- puntosX = [] Lista con los puntos en X de las piezas
- puntosY = [] Lista con los puntos en Y de las piezas

En los dos primeros vectores se guarda los recursos disponibles y requerimientos de producción, en los siguientes se crean vectores que contenga las posiciones finales de las esquinas de cada uno de los cortes que tiene que realizar el brazo robótico UR3.

Paso 2: Se inicia la comunicación TCP/IP con Python como servidor y el robot como cliente. Se crea dos tuplas que representan la conexión tanto para la recepción y el envío de datos. La primera tupla contiene la dirección IP del cliente y el puerto de conexión que es "44444" y la segunda tiene la misma dirección IP con el puerto "30002" que está abierto para *Script Programming* en el robot.

Paso 3: Una vez establecida la comunicación comienza la recepción de datos. En primera instancia, se debe ingresar los recursos disponibles y guardarlos en el vector *datT*. Luego, se crea un bucle de recepción donde se concatena todos los requerimientos de cada una de las piezas que se desea cortar y se guarda en el vector *datP*.

Posteriormente, se asignan los datos a variables, se busca la coordenada de la esquina inferior izquierda de la plancha para que el algoritmo de corte tenga referencia y se configura el ángulo de giro para poder resolver las dos posibles orientaciones, horizontal o vertical, en las que se puede configurar la plancha.

Paso 4: Una vez este proceso se ha realizado se da paso al algoritmo de corte el cual da como resultado cinco pares ordenados para cada una de las piezas, los cuales representan las esquinas repitiéndose el primero ya que el robot tiene que regresar al punto desde el que parte. La función *greedypacker.BinManager()* permite elegir el tipo de algoritmo de corte en este caso "*guillotine*" y la heurística "*best_longside*".

El conjunto de pares ordenados tiene que ser rotado y trasladado hacia la posición y orientación de la plancha real lo que se consigue programando las ecuaciones (2) y (3) previamente explicadas.

Paso 5: Para poder enviar los datos hacia el robot se crea un nuevo socket de envió el cual se configura con la tupla para *Script Programming*. Esto permite que Python se conecte directamente con el robot y pueda programar su movimiento sin la necesidad de la interfaz PolyScope. Para esto dentro de Python se programó dos comandos "movel" con los argumentos requeridos que harán líneas de corte rectas al momento de formar los rectángulos. Por último, se cierra el socket para terminar la comunicación y el robot se pone en marcha.

D. Digital Twin (Puesta en Funcionamiento)

Para poner en funcionamiento el *Digital Twin* del brazo robótico UR3 en el marcado de piezas para posterior corte se configura el entorno virtual dentro de *Experior* y la máquina virtual URsim de *Universal Robot*.

En primer lugar, se ubica el brazo robótico UR3 en el espacio de trabajo. Para configurar el mismo, en la pestaña *Propierties* en la sección *Position* se asigna la coordenada (-600;905;-600) [mm] que representa el lugar espacial donde está ubicada la base del robot, y en la sección *Robot Communication* se asigna la dirección IP del robot real. Para terminar la configuración del brazo robótico se selecciona el *Gripper* y se conecta al efector final del robot.

Para configurar la plancha se ingresa a la pestaña *Communication* y se crea una unidad de Modbus con la dirección IP del robot de donde se va a recibir la información (posición y rotación). Una vez creada esta unidad, dentro de las configuraciones de la plancha se forma un inciso donde se ingresan las direcciones de Modbus creadas previamente en PolyScope. A continuación, se ingresan todas las configuraciones de acuerdo con lo explicado en la sección anterior. En la Fig.17 se presenta el entorno de *Experior* configurado.

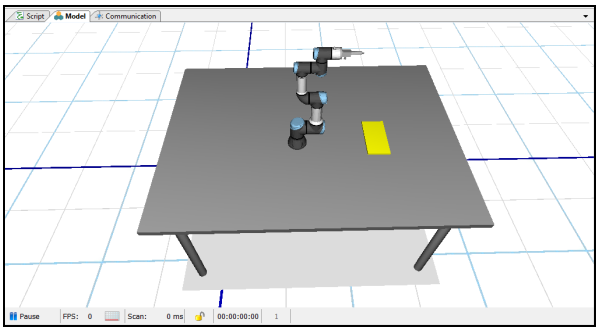


Fig.17. Entorno configurado dentro de Experior

Para establecer la comunicación y trasmitir la información necesaria entre URsim (ubicada en el sistema operativo Ubuntu, sobre una máquina virtual) y *Experior* (instalado en Windows) es necesario relacionar las dos redes mediante un Puente de Red.

Una vez que inicia la máquina virtual en PolyScope se ejecuta el programa, el cual espera hasta que exista comunicación con el servidor por lo que es necesario dar inicio al programa desarrollado en Python. A continuación, PolyScope muestra los cuadros de entrada donde se ingresa los diferentes datos como: el tamaño de la plancha, número de piezas a cortar y las dimensiones de las piezas. Terminado el ingreso, el algoritmo de corte genera los puntos y envía comandos de movimiento que pondrán en funcionamiento al robot junto con el *Digital Twin* hasta finalizar los cortes y así terminar con todo el proceso.

III. RESULTADOS

A. Resultado del algoritmo de corte

Se plantea un problema en el cual la suma del área de las piezas sea igual al de la plancha. La tabla 2 muestra el tamaño del recurso disponible además de las dimensiones de las piezas a ser cortadas.

Tabla.2. Datos de ingreso al algoritmo

Datos ingresados		
Ítem	Dimensiones (mm)	
	Width	Length
Plancha	300	150
Pieza 1	100	100
Pieza 2	100	50
Pieza 3	50	150
Pieza 4	50	50
Pieza 5	50	100
Pieza 6	50	150
Pieza 7	50	75
Pieza 8	50	75

El algoritmo de corte guillotina resuelve el problema ubicando todas las piezas dentro de la plancha como muestra gráficamente la Fig.18, esto representa una solución adecuada para nuestro caso de estudio.

REVISTA PERSPECTIVAS
e-ISSN: 2661-6688

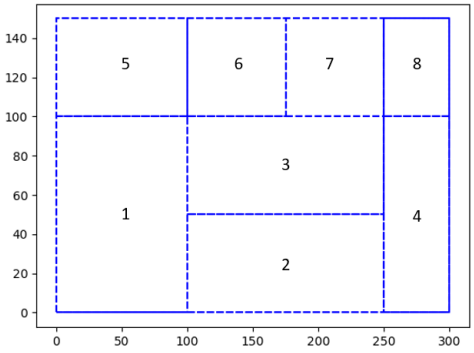


Fig.18. Respuesta grafica del algoritmo implementado

Los resultados gráficos mostrados a continuación en la Fig.19 corresponden a la solución del mismo problema planteado para tres diferentes algoritmos. A partir de estos resultados se puede inferir que los algoritmos *Shelf* y *Skyline* no logran ubicar todas las piezas debido a su lógica de programación. Por otro lado, el algoritmo de *Maximal Rectangles* llega hacia una solución óptima ya que se trata de un proceso complejo al igual que su programación por ende tiene un costo computacional alto. Debido a esto se ratifica la correcta selección del algoritmo *Guillotine,* el cual entrega buenos resultados a un costo computacional bajo.

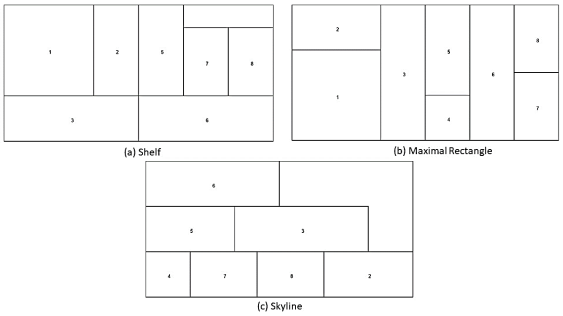


Fig.19 (a) Resultado gráfico del algoritmo *Shelf*. (b) Resultado gráfico del algoritmo *Maximal Rectangle*. (c) Resultado gráfico del algoritmo *Skyline*.

B. *Pruebas de rotación y traslación*

Las pruebas desarrolladas a continuación evalúan las ecuaciones planteadas en la sección anterior.

Continuando con el ejemplo anterior se añade los datos recabados del entorno de trabajo mediante la Wrist Camera. La ubicación del centro de la pieza es (200,200) [mm] correspondientes a las coordenadas X e Y del sistema de referencia de la base del robot mientras que el eje Z se estima dependiendo del grosor de la plancha, a la cual se le ha asignado 4 [mm]. La rotación es $\pi/4$ [rad] y para las dimensiones de la plancha se considera que el ancho es mayor que el largo. De esta forma la variable que guarda la cámara es:

$$object\_location = [200, 200, 4, 0.785, 0.785, 0.000\ ]$$

Los resultados gráficos mostrados en la Fig.20 son los devueltos por el algoritmo utilizando los datos del ambiente descritos. Como se observa, la figura roja representa la distribución de piezas ubicadas y rotadas conforme a los datos recaudados del ambiente de trabajo.

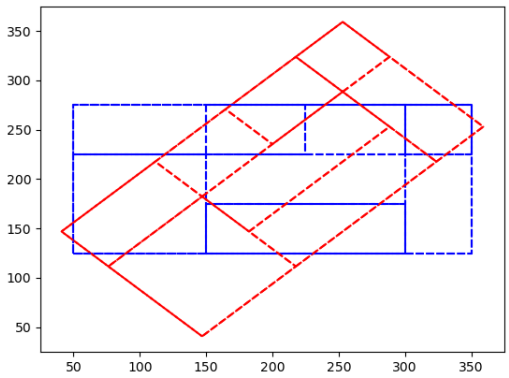
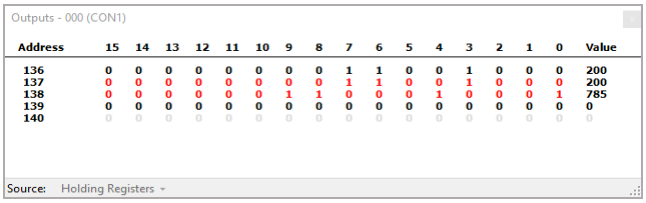


Fig.20. Resultado gráfico de la ubicación de las piezas respecto al entorno.

C. *Comunicación entre PolyScope y Experior.*

El tráfico existente se comprueba en la ventana comunicaciones la cual muestra los paquetes de *Holding Registers* que son las entradas de las configuraciones de la pieza que corresponden a las salidas Modbus en *PolyScope*. La Fig.21 muestra en código binario el valor del centro de la pieza y el ángulo de giro enviado desde PolyScope, comprobando así la correcta comunicación entre estas dos plataformas.



Fig,21. Datos de conectividad del ángulo y la ubicación del centro de la pieza

Para calcular el tiempo de latencia presente en la comunicación tanto en la adaptación del entorno como en el movimiento del *Digital Twin* se realiza una prueba enviando paquetes de datos desde *PolyScope* hacia *Experior* y almacenando el tiempo que tarda el mismo en responder. Se consideró una muestra de 40 datos la cual se estima suficiente para representar el comportamiento del sistema. Se requiere conocer la tendencia central y la dispersión de los datos obtenidos por lo que se utiliza estadística descriptiva, como se muestra en la tabla 3

Tabla.3. . Estadística descriptiva de los datos de tiempos de retardo

Análisis de datos	
Tamaño de la muestra	40
Retardo Mínimo [ms]	2,72
Retardo Máximo [ms]	6,9
Amplitud [ms]	4.18
Media [ms]	3,64
Moda [ms]	2,83
Desviación Estándar [ms]	1,1434

Después del análisis se obtuvo una media de 3.64 [ms] que muestra la tendencia central a la que convergen los datos de la muestra que va desde el tiempo de retardo mínimo, 2.72 [ms], hasta el máximo, 6.9 [ms]. Se observa también la dispersión de estos datos con respecto a dicha media, que, en el caso planteado, es de 1.1434 [ms]. Con estos datos se

concluye que el retardo existente entre la interfaz gráfica URsim y la réplica de movimientos del *Digital Twin* es imperceptible visualmente. Cabe recalcar que se trabaja únicamente en entornos virtuales por lo que el tiempo de latencia es solo el que tarda en enviar y recibir datos *Experior*.

D. Resultado gráfico de la adaptación al entorno

Al comprobar la correcta comunicación y un fidedigno envió como recepción de registros. Dentro de *Experior* debe reflejarse una correcta adaptación al medio. La Fig.22 muestra el resultado gráfico que genera *Experior* para el último ejemplo planteado, el cual ubica la pieza en la coordenada (200,200) [mm] desde el sistema de referencia de la base del robot y rotada un ángulo de 45°.

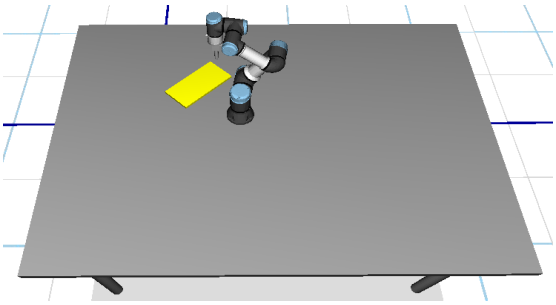


Fig.22. Respuesta grafica de Experior ubicando el centro en (200, 200) [mm] y rotado 45°

E. Fidelidad del Digital Twin en el proceso de marcado para corte de superficies

La fidelidad va a ser probada con un caso de estudio donde se ubican tres piezas que cubran el área total de la plancha. La tabla 3 muestra los recursos disponibles y la orden de producción utilizadas. Los datos del ambiente son los siguientes:

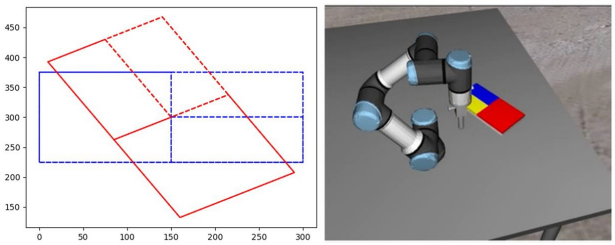
object_location = [150,300,4,-0.523,-0.523,0.000]

Para la respuesta gráfica en *Experior* se configura el entorno de tal manera que las piezas marcadas tomen un color diferente al de la plancha. Así, se comprueba gráficamente que el robot se mueva de forma adecuada con las configuraciones del ambiente.

Los resultados numéricos de las coordenadas de cada pieza obtenidos por el algoritmo de corte generados en Python presentan un error con los datos reales de la ubicación del robot en las coordenadas que realiza el corte ya que estos están limitados por la exactitud y precisión mecánicas del robot. Los datos reales de la tabla 4 generan un error respecto a los calculados de:

TABLA.4. DATOS DE INGRESO PARA PRUEBAS DE FIDELIDAD

Datos Ingresados		
Ítem	Dimensiones (mm)	
	Width	Length
Plancha	300	150
Pieza 1	75	150
Pieza 2	75	150
Pieza 3	150	150



(a) (b)

Fig.23. (a) Respuesta gráfica del algoritmo de corte con giro de -30°. (b) Resultado gráfico de Experior después del corte.

RMSE= 0.242%

IV. CONCLUSIONES

Una interfaz *Digital Twin* apoya la actividad de supervisión del operador de acuerdo con los requisitos de los sistemas de fabricación flexible en el proceso de corte de superficies con una arquitectura HCMI que permite la interacción entre máquina-computador-humano. El uso de la *Wrist Camera* y el software *Camera Locate* como una adaptación para el brazo robótico UR3 permitió una percepción del ambiente de trabajo adecuada obteniendo la variable *object_location* que resulto de gran ayuda para la programación dentro de *PolyScope* y *Python*. Además, la elección del algoritmo *Guillotina* tubo resultados favorables frente a varios casos de estudio ubicando las piezas con mejor disposición que los algoritmos *Shelf* y *Skyline*, y presenta resultados similares con el algoritmo *Maximal Rectangles*. Sin embargo, el algoritmo *Guillotina* elegido posee una programación menos compleja por lo que su costo computacional es reducido. Los resultados obtenidos confirmaron que la interfaz Digital Twin permite una supervisión casi en tiempo real con una media de retardo de 3.64 [ms] y una fidelidad al momento de seguir los segmentos de corte dados por el algoritmo de 99,758%.

AGRADECIMIENTOS

Los autores desean agradecer a la Corporación Ecuatoriana para el Desarrollo de la Investigación y Academia- CEDIA por su contribución a la innovación, a través de los proyectos CEPRA, en especial por el proyecto CEPRA-XIV-2020-08-RVA "Tecnologías Inmersivas Multi-Usuario Orientadas a Sistemas Sinérgicos de Enseñanza-Aprendizaje". También, un agradecimiento a la Escuela Superior Politécnica de Chimborazo - ESPOCH y al Grupo de Investigación GITEA por su aporte para el desarrollo de este trabajo.

REFERENCIAS

[1] R. Rosen, G. Von Wichert, G. Lo, and K. D. Bettenhausen, "About the importance of autonomy and digital twins for the future of manufacturing," *IFAC-PapersOnLine*, vol. 28, no. 3, pp. 567–572, May 2015, doi: 10.1016/j.ifacol.2015.06.141.

[2] R. Y. Zhong, X. Xu, E. Klotz, and S. T. Newman, "Intelligent Manufacturing in the Context of Industry 4.0: A Review," *Engineering*, vol. 3, no. 5, pp. 616–630, Oct. 2017, doi: 10.1016/J.ENG.2017.05.015.

[3] R. Carmen, L. José, C. Cabrera, and L. Renato, "Creación de ambientes virtuales inmersivos con software libre," *Rev. Digit. Univ.*, vol. 8, no. 6, pp. 3–7, 2007.

[4] A. J. Matteo and E. Coto, "Una herramienta para generar Mundos

REVISTA PERSPECTIVAS
e-ISSN: 2661-6688

Virtuales Inmersivos View project SOS Telemedicina View project," Caracas, 2000. Accessed: Jun. 04, 2020. [Online]. Available: https://www.researchgate.net/publication/228858236.

[5] D. Enríquez, J. J. Arellano Pimente, M. Á. Hernández López, and O. S. Nieva García, "Didactic use of immersive virtual reality with NUI focused on the inspection of wind turbines," *Apertura*, vol. 9, no. 2, pp. 8–23, Oct. 2017, doi: 10.32870/ap.v9n2.1049.

[6] L. Pérez, S. Rodríguez-Jiménez, N. Rodríguez, R. Usamentiaga, and D. F. García, "Digital twin and virtual reality based methodology for multi-robot manufacturing cell commissioning," *Appl. Sci.*, vol. 10, no. 10, 2020, doi: 10.3390/app10103633.

[7] M. L. S. Wallace J. Hopp, *Factory Physics*. 2011.

[8] J. Lee, H.-A. Kao, and S. Yang, "ScienceDirect Service innovation and smart analytics for Industry 4.0 and big data environment," *Procedia CIRP*, vol. 16, pp. 3–8, 2014, doi: 10.1016/j.procir.2014.02.001.

[9] J. Daniel, H. Edgar, S. Cespedes, D. Andres, G. David, and L. Fumagalli, "Human-Computer-Machine Interaction for the Supervision of Flexible Manufacturing Systems : A Case Study," *IFAC-PapersOnLineC*, vol. 7, pp. 1–6, 2018.

[10] J. Lee, E. Lapira, B. Bagheri, and H. an Kao, "Recent advances and trends in predictive manufacturing systems in big data environment," *Manuf. Lett.*, vol. 1, no. 1, pp. 38–41, 2013, doi: 10.1016/j.mfglet.2013.09.005.

[11] A. Parrott and L. Warshaw, "Industry 4.0 and the digital twin," *Deloitte Univ. Press*, vol. 5, no. 2, pp. 1–17, 2017, [Online]. Available: https://dupress.deloitte.com/dup-us-en/focus/industry-4-0/digital-twin-technology-smart-factory.html.

[12] C. Koulamas and A. Kalogeras, "Cyber-Physical Systems and Digital Twins in the Industrial Internet of Things," *IEEE Software*, pp. 1–4, 2018.