DRM – 09/2022

Universidad Católica Boliviana "San Pablo"
Facultad de Ingeniería
DIPLOMADO EN ROBÓTICA (2DA VERSIÓN)

ARTÍCULO DE INVESTIGACIÓN EN FORMATO IEEE
PRESENTADO A LA FACULTAD
DE INGENIERÍA PARA LA OBTENCIÓN DEL GRADO ACADÉMICO
DEL DIPLOMADO EN ROBÓTICA (2DA VERSIÓN)

SIMULACIÓN EDUCATIVA DE KUKA KR6 BASADA EN KUKA SMARTPAD

Realizado por:

WILBER ÁLVARO ROJAS FERNÁNDEZ

2022

Simulación Educativa de KUKA KR6 basada en KUKA smartPAD

Documento para optar por el Título de *Diplomado en Robótica*
Universidad Católica Boliviana San Pablo Sede La Paz

Wilber Álvaro Rojas Fernández
Postulante al Título del Diplomado en Robótica

Fabio Richard Díaz Palacios
Profesor del Módulo Final del Diplomado en Robótica

***Resumen*—Los brazos robóticos son cada vez más utilizados en las industrias, no obstante, la educación enfocada a enseñar el uso y la programación de estos es muy limitada. La adquisición de un brazo robótico industrial es tan costoso que las instituciones solo pueden disponer de un numero limitado de robots para su educación. Si bien existen simuladores que cada marca pone a disposición, estas por lo general, son de paga en base a suscripciones. Por este motivo, el presente proyecto tiene como objetivo desarrollar un simulador enfocado a la programación de robots KUKA por medio de su control KCP. Se utilizo el sistema operativo Ubuntu 20.04, así como el framework de ROS Noetic. Se simulo específicamente el brazo KUKA "KR 6 R900 sixx" y se desarrollo la simulación para el control KUKA smartPAD. Asimismo, se programo una pizarra virtual en un entorno simulado del laboratorio C201 de la universidad.**

***Palabras clave*—KUKA Robot, KUKA smartPAD, Simulación, ROS, RVIZ.**

***Línea de investigación:* Ciencia, Tecnología e Innovación; Automatización y Robótica: Incluyendo el estudio de vehículos autónomos no tripulados aéreos, terrestres u otros, la aplicación de técnicas de mapeo y localización, sistema operativo para robots, simulación de agentes robóticos en distintos entornos.**

I. Introducción

En los últimos años la implementación de brazos robóticos en áreas industriales se a incrementado. Fabricas de autos, alimentos, cemento, medicina, entre otros, utilizan brazos robóticos para mejorar su productividad. Esta tecnología mejora los sistemas de automatización, puede realizar trabajos repetitivos con precisión y a un menor costo [1].

Por otro lado, los brazos robóticos han demostrado ser altamente peligrosos, capaces de causar lesiones graves o incluso la muerte de personas durante un descuido. A causa de esto, la educación enfocada a la enseñanza de brazos robóticos enfatiza la importancia de la seguridad respecto a su uso [2].

No obstante, la educación muchas veces no puede financiar los altos costos de esta tecnología. Lo que provoca en consecuencia que los estudiantes tengan horas limitadas de aprendizaje con el robot real. Empresas como KUKA desarrollaron simuladores para sus robots. El problema que surge es que la adquisición del simulador oficial es por suscripción. Por lo tanto, el presente proyecto busca desarrollar un simulador de código abierto basado en la tecnología de KUKA.

Este trabajo fue entregado para su revisión en fecha 30 de Septiembre de 2022. El autor declara que el trabajo es de su completa autoría y referencia adecuadamente otros trabajos.

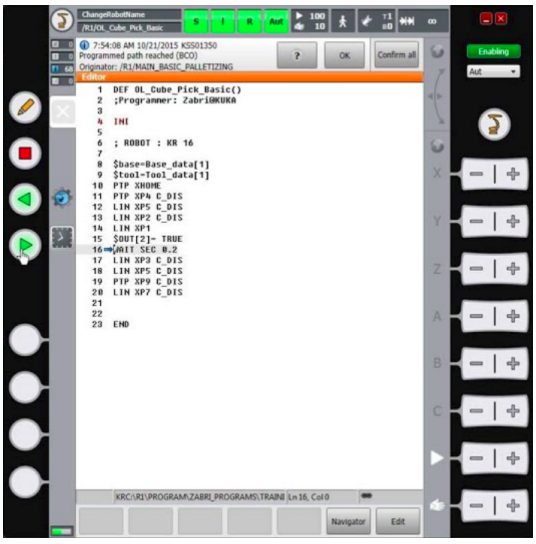

Figura 1. Programa de simulación de KUKA.

II. Objetivos

A. Objetivo General

Desarrollar un sistema educativo basado en KUKA smartPAD para la programación de brazos robóticos.

B. Objetivos Específicos

- Manipular el robot por movimiento de Ejes y coordenadas Cartesianas.
- Emplear un historial de posiciones objetivo para el movimiento del robot.
- Implementar tipos de movimiento PTP y LIN.
- Diseñar una interfaz para una programación basada en KUKA smartPAD.
- Desarrollar un pizarra virtual editable por los movimientos del robot.

III. Materiales y Métodos

A. *Simulación en ROS*

El proyecto se desarrollo con el sistema operativo Ubuntu 20.04 y el framework de ROS con la distribución Noetic. El modelo del robot "KR 6 R900 sixx" se obtuvo del repositorio de *ROS Industrial* [3].

Se genero un *Workspace* donde se duplico los *packages* para la simulación del KR6. El *package* "kuka_kr6_support" contiene los archivos 3D de las partes del robot, además, contiene archivos URDF que indican la relación de posición, colisión y movimiento que tienen estas partes del robot. Respecto al "kuka_resources", este *package* contiene configuraciones globales para todos los robots, como colores y texturas.

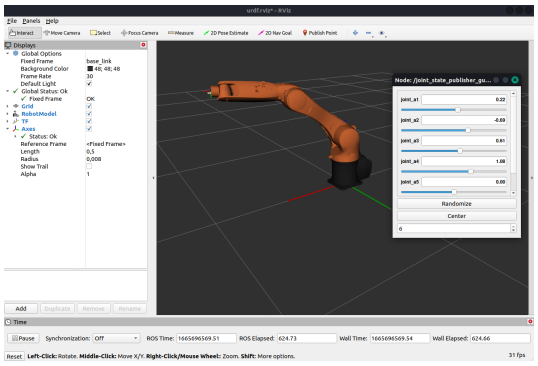

Figura 2. Simulación básica en RVIZ.

En la imagen 2 se muestra una simulación básica del robot, con la que se verifica que los archivos 3D y URDF están correctamente configurados. Se creo un *package* que contiene un archivo launch, el cual carga el URDF del robot al simulador de RVIZ. Solo para verificar el correcto funcionamiento de los movimientos del robot, el archivo launch también corre una interfaz gráfica que publica los movimientos que debe realizar cada *joint*.

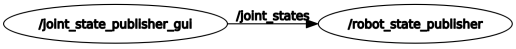

Figura 3. Gráfica de Nodos: GUI de RVIZ.

La comunicación entre la interfaz gráfica y el robot simulado en RVIZ se realiza por el tópico de "joint_states", como se muestra en la figura 3.

Todos los *packages* de la simulación de la Figura 2 se pueden encontrar en el Anexo 1. Asimismo, el comando para ejecutar la simulación es el siguiente:

```
roslaunch basic_simulation kr6.launch
```

B. *MoveIt*

MoveIt [4] permite una manipulación más completa de los brazos robóticos por medio de una interfaz en RVIZ. Hace posible dar una posición objetivo, con la cual, calcula y traza los movimientos para llegar al objetivo dado. Asimismo, MoveIt provee librerías de Python y C++, lo que hace posible manipular el robot por medio de scripts.

Para generar un *package* de MoveIt capaz de correr el KUKA KR6, se utilizó el asistente de configuración, el cual se ejecuta con el siguiente comando:

```
roslaunch moveit_setup_assistant
↪  setup_assistant.launch
```

Se despliega la ventana del asistente de configuración, en donde se debe cargar el archivo URDF, como se muestra en la Figura 4.

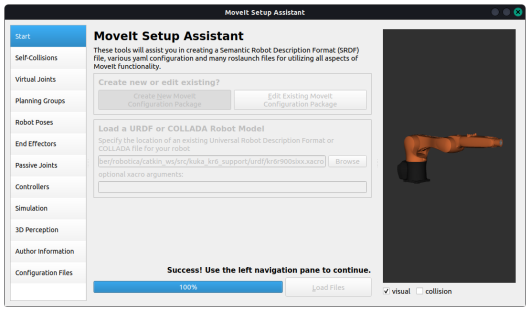

Figura 4. Configuración de MoveIt: Cargar URDF.

Luego se genero la matriz de colisión con una densidad de muestreo de 10000 puntos, que es el valor recomendado por MoveIt. La densidad de muestreo especifica cuántas posiciones de robot aleatorias se deben verificar para detectar autocolisiones. El resultado se puede observar en la Figura 5.

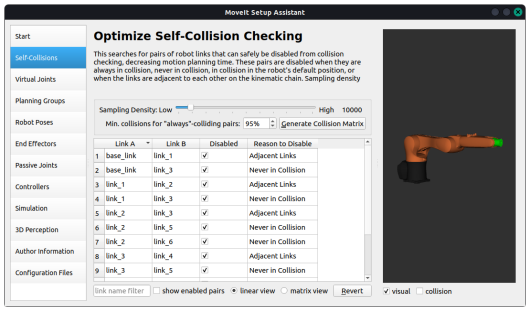

Figura 5. Configuración de MoveIt: Matriz de colisión.

Para asegurar que el robot se simule correctamente, es decir, con las coordenadas de la base en (0,0,0). Se creo una unión virtual entre la base del robot y el mundo, la configuración se la puede observar en la Figura 6.

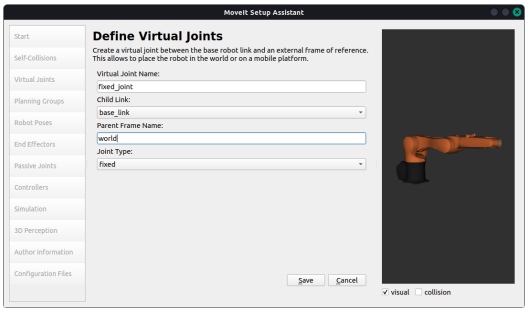

Figura 6. Configuración de MoveIt: Virtual Joints.

Se requiere definir grupos de planeación para registrar la relación entre las partes de un robot. Por ejemplo, que piezas corresponden a que brazo, pierna, etc. En este caso, todas las piezas corresponden a un único brazo robótico. En la Figura 7 se muestra este proceso, en donde el solucionador de cinemática inversa se lo asigno al algoritmo de KDL.

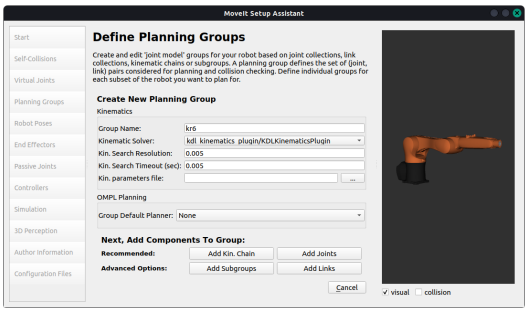

Figura 7. Configuración de MoveIt: Grupos de Planeación.

El algoritmo de KDL requiere que se defina una cadena cinemática entre el primer y ultimo link, También llamados “Base Link” y “Tip Link”.

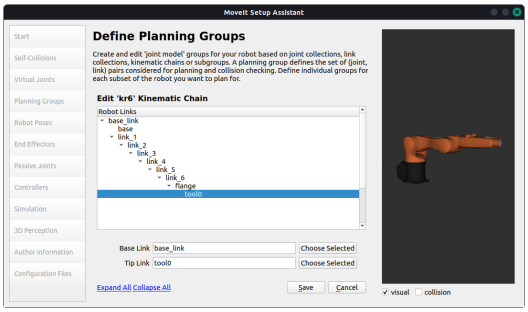

Figura 8. Configuración de MoveIt: Cadena Cinemática.

Por ultimo, se genero el package con todas las configuraciones realizadas. El Anexo 2 contiene una demostración de RVIZ con MoveIt manipulando el robot KUKA KR6 y todos los packages correspondientes para su simulación.

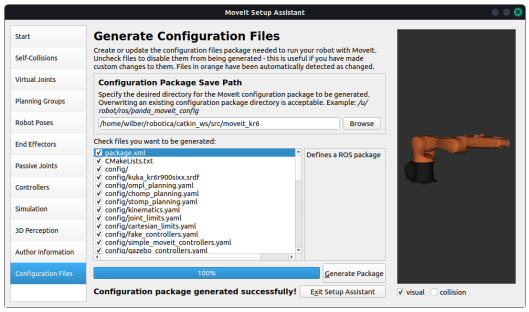

Figura 9. Configuración de MoveIt: Archivos de configuración.

IV. Resultados

Con la adición de MoveIt a la simulación del KUKA KR6, se puede iniciar con el desarrollo de la simulación del KUKA smartPAD. Este control permite manipular al robot mediante movimientos por ejes y coordenadas mundo, también conocidas como coordenadas cartesianas. Ambos tipos de movimientos son ejecutados por botones “**+**” y “**-**”, indicando la dirección del movimiento. Aclarar que también existe una configuración de velocidad que es manipulada por botones similares. En la imagen 10, se puede observar la configuración estética de esos botones.

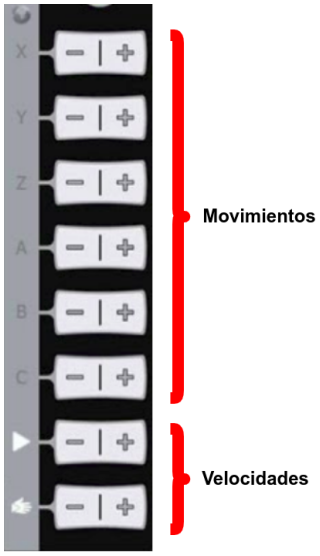

Figura 10. KUKA smartPAD: botones de movimientos y velocidades.

Para integrar a la simulación de ROS los movimientos por ejes y coordenadas cartesianas semejante al funcionamiento del KUKA smartPAD, se requiere el uso de las librerías del framework MoveIt.

Se desarrollo un programa de Python que se conecta con la simulación del robot KUKA KR6 que fue configurado con el asistente de MoveIt. Este programa obtiene los valores actuales de los joints y puede modificarlos mediante la interfaz que se muestra en la Figura 11. Asimismo, se añadió los botones para el control de la velocidad del robot.

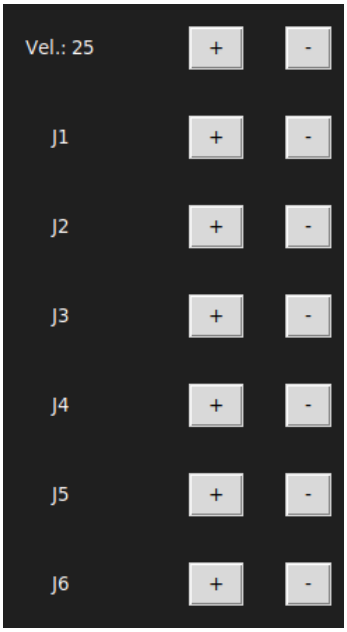

Figura 11. ROS smartPAD: Botones de Movimiento de ejes.

De igual forma, el programa cuenta con un cálculo de la ruta cartesiana, que se calcula respecto al ultimo link del robot. Esto permite mover la herramienta del robot en un solo eje cartesiano, por lo que se lo integro en una función que es llamada por los botones de la interfaz de la Figura 12. El botón "Home" adicional tiene la función de mover el robot a la posición inicial.

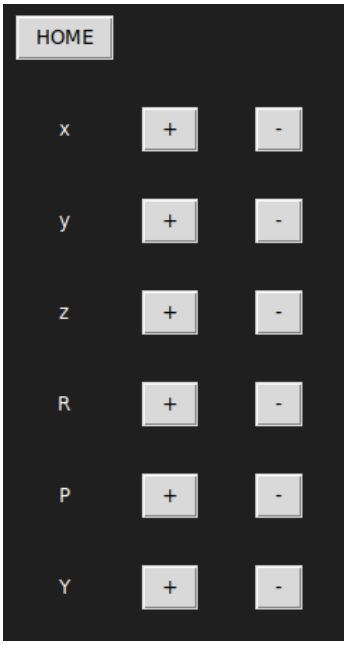

Figura 12. ROS smartPAD: Botones de Movimiento cartesiano.

A. *Secuencia de Movimiento*

El control KUKA smartPAD tiene la función de guardar puntos PTP y LIN, para luego correrlos en una secuencia. En la simulación desarrollada, un punto PTP se ejecuta con los movimientos de los ejes, mientras que un punto LIN se ejecuta con los movimientos cartesianos.

Se utilizó listas de Python para almacenar las diferentes posiciones del robot en puntos. Como se muestra en la Figura 13, cada punto tiene asignado un movimiento PTP o LIN. El control de ROS smartPAD desarrollado, tiene la opción de guardar, borrar, o mover el robot a un punto.

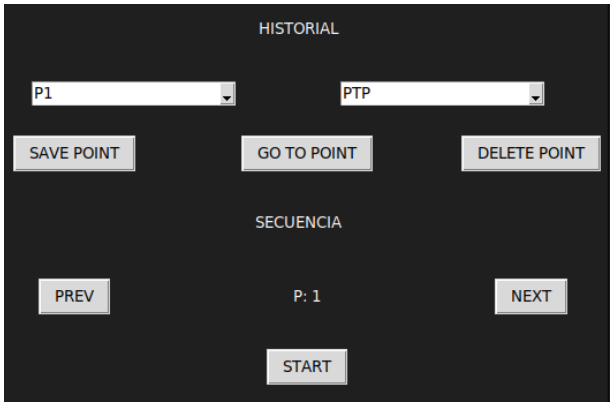

Figura 13. ROS smartPAD: Botones de Movimiento cartesiano.

Respecto a la secuencia, ROS smartPAD cuenta con un botón de inicio, el cual, ejecuta todo los puntos guardados de manera ordenada. Adicionalmente se cuenta con botones para mover el robot al siguiente o anterior punto.

B. *Pizarra Virtual*

Todo lo que se ha desarrollado previo a este punto, trata de seguir el funcionamiento del KUKA smartPAD. No obstante, existen ejercicios educativos que también es importante replicar. Por este motivo, se ha desarrollado una pizarra virtual, en la que el robot puede escribir solo si esta cerca de una mesa que se simula igualmente en RVIZ.

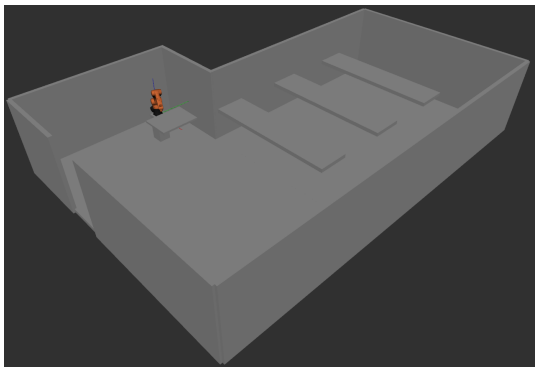

Figura 14. RVIZ: Laboratorio c201

Como primer paso, se creo un entorno similar al laboratorio c201. En caso de que algún ejercicio requiera o no de la mesa, esta puede ser adicionada o removida por medio de botones del ROS smartPAD.

Para determinar si el robot esta "rayando" en la pizarra, se implemento cinemática directa enfocada en calcular la posición espacial del extremo del KR6.

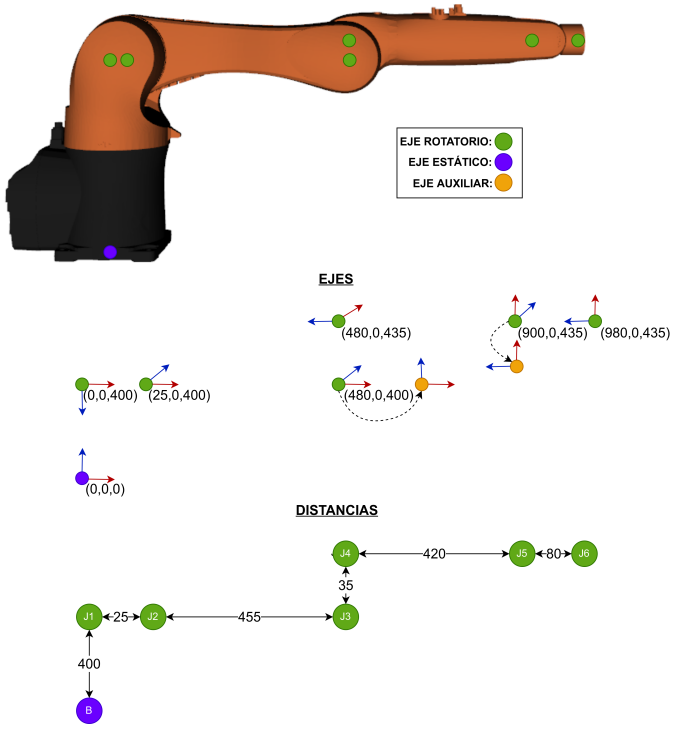

Figura 15. Análisis por Denavit-Hartenberg.

La Figura 15, muestra el análisis de ejes por el método de Denavit-Hartenberg. En donde las unidades están en milímetros. La matriz resultante se muestra en la Tabla I.

Tabla I
MATRIZ DH.

θ	**d**	**a**	α
0	400	0	pi
j1	0	25	pi/2
j2	0	455	0
j3	0	0	pi/2
pi/2	35	0	-pi/2
j4-pi/2	-420	0	-pi/2
j5	0	0	pi/2
0	-80	0	0

Se puede observar la representación simbólica de matrices de la ecuación 1 a la ecuación 8.

$$T_{01} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0{,}400 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1}$$

$$T_{12} = \begin{bmatrix} cos(j1) & 0 & sin(j1) & 0{,}025 * cos(j1) \\ sin(j1) & 0 & -cos(j1) & 0{,}025 * sin(j1) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2}$$

$$T_{23} = \begin{bmatrix} cos(j2) & -sin(j2) & 0 & 0{,}455 * cos(j2) \\ sin(j2) & cos(j2) & 0 & 0{,}455 * sin(j2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

$$T_{34} = \begin{bmatrix} cos(j3) & 0 & sin(j3) & 0 \\ sin(j3) & 0 & -cos(j3) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4}$$

$$T_{45} = \begin{bmatrix} 0) & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0{,}035 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

$$T_{56} = \begin{bmatrix} sin(j4) & 0 & cos(j4) & 0 \\ -cos(j4) & 0 & sin(j4) & 0 \\ 0 & -1 & 0 & -0{,}42 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6}$$

$$T_{67} = \begin{bmatrix} cos(j5) & 0 & sin(j5) & 0 \\ sin(j5) & 0 & -cos(j5) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{7}$$

$$T_{78} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -0{,}08 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8}$$

$$T_{08} = T_{01} \cdot T_{12} \cdot T_{23} \cdot T_{34} \cdot T_{45} \cdot T_{56} \cdot T_{67} \cdot T_{78} \tag{9}$$

El resultado de la multiplicación de en la ecuación (9) es una matriz que contiene los datos de la posición del extremo del robot:

$$\begin{bmatrix} T_{08_{03}} \\ T_{08_{13}} \\ T_{08_{23}} \end{bmatrix} = \begin{bmatrix} px \\ py \\ pz \end{bmatrix} \tag{10}$$

En 10 [px; py; pz] definen su valor en base a las siguientes ecuaciones:

$$
\begin{aligned}
px \quad = \quad & -0{,}08 * (-(-sin(j2) * cos(j1) * cos(j3) \\
& -sin(j3) * cos(j1) * cos(j2)) * cos(j4) \\
& +sin(j1) * sin(j4)) * sin(j5) \\
& +0{,}08 * (-sin(j2) * sin(j3) * cos(j1) \\
& +0{,}035 * sin(j2) * cos(j1) * cos(j3) \\
& +0{,}035 * sin(j3) * cos(j1) * cos(j2) \\
& +0{,}42 * cos(j1) * cos(j2) * cos(j3) \\
& +0{,}455 * cos(j1) * cos(j2) + 0{,}025 * cos(j1)
\end{aligned}
$$

$$
\begin{aligned}
py \quad = \quad & -0{,}08 * (-(sin(j1) * sin(j2) * cos(j3) \\
& +sin(j1) * sin(j3) * cos(j2)) * cos(j4) \\
& +sin(j4) * cos(j1)) * sin(j5) \\
& +0{,}08 * (sin(j1) * sin(j2) * sin(j3) \\
& -sin(j1) * cos(j2) * cos(j3)) * cos(j5) \\
& +0{,}42 * sin(j1) * sin(j2) * sin(j3) \\
& -0{,}035 * sin(j1) * sin(j2) * cos(j3) \\
& -0{,}035 * sin(j1) * sin(j3) * cos(j2) \\
& -0{,}42 * sin(j1) * cos(j2) * cos(j3) \\
& -0{,}455 * sin(j1) * cos(j2) - 0{,}025 * sin(j1)
\end{aligned}
$$

$$
\begin{aligned}
pz \quad = \quad & 0{,}08 * (sin(j2) * sin(j3) \\
& -cos(j2) * cos(j3)) * sin(j5) * cos(j4) \\
& +0{,}08 * (-sin(j2) * cos(j3) \\
& -sin(j3) * cos(j2)) * cos(j5) \\
& -0{,}035 * sin(j2) * sin(j3) \\
& -0{,}42 * sin(j2) * cos(j3) \\
& -0{,}455 * sin(j2) - 0{,}42 * sin(j3) * cos(j2) \\
& +0{,}035 * cos(j2) * cos(j3) + 0{,}4
\end{aligned}
$$

Estas ecuaciones son base para el funcionamiento de la pizarra virtual. Se creo un nuevo programa de Python que publica una imagen generada con la librería de OpenCV por un tópico hacia RVIZ, como se muestra en la Figura 16. El programa se suscribe al tópico de “joint_states”, se aplica las ecuaciones de 10 y se obtiene la posición espacial del efector final del robot. El robot puede realizar dibujos en la pizarra, si mueve el extremo del KUKA sobre la mesa simulada en RVIZ.

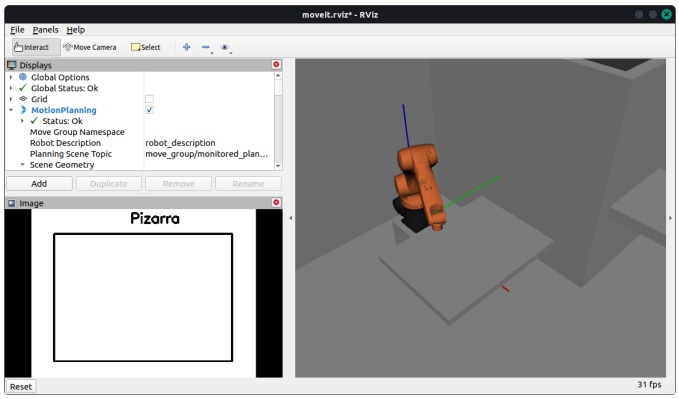

Figura 16. RVIZ: Pizarra

En la figura 17 se muestra la diferencia entre una secuencia con todos los puntos PTP y otra donde todos los puntos son LIN.

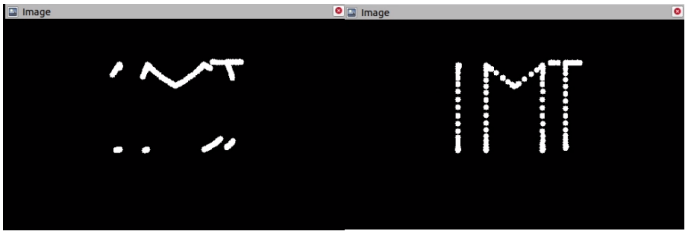

Figura 17. Comparación: PTP vs LIN

V. Recomendaciones

Para una simulación mas ligera, de ser posible, se recomienda el uso del simulador de RVIZ en lugar de Gazebo. De esta manera, los requerimientos computacionales se disminuyen. Por otro lado, si otro programa requiere la posición del robot, se recomienda la implementación de cinemática inversa para no saturar las librerías de MoveIt.

VI. Conclusiones

En conclusión, se logro manipular el robot por movimiento de ejes y coordenadas cartesianas. Se implemento efectivamente un historial de puntos que contempla los movimientos PTP y LIN. Se desarrollo una interfaz simplificada a la del KUKA smartPAD, ya que no se contempla generación de código en la interfaz de ROS smartPAD. La pizarra virtual permite que el robot simule una trayectoria de dibujo, no obstante, el dibujo se realiza por puntos y su continuidad depende de la información del tópico “joint_states”.

ANEXOS

Los siguientes anexos se encuentran en el mismo repositorio de Github, para una mejor revisión se ha separado los puntos de interés en los siguientes enlaces:

1. Simulación básica: https://github.com/WilberRojas/KUKA-smartPAD-ROS/tree/basic-simulation
2. Integración de MoveIt: https://github.com/WilberRojas/KUKA-smartPAD-ROS/tree/moveit-integration
3. Simulación con KUKA smartPAD: https://github.com/WilberRojas/KUKA-smartPAD-ROS

REFERENCIAS

[1] M. Behdad, "Socialization of industrial robots: An innovative solution to improve productivity."

[2] R. Bogue, "Robotic vision boosts automotive industry quality and productivity."

[3] ROS Industrial, "Kuka experimental," Disponible en: https://github.com/ros-industrial/kuka_experimental.

[4] MoveIt, "Moving robots into the future," Disponible en: https://moveit.ros.org/.

Wilber Álvaro Rojas Fernández
Ingeniero Mecatrónico