



## **ESCUELA SUPERIOR POLITÉCNICA DEL LITORAL**

### **Facultad de Ingeniería en Electricidad y Computación**

“DESARROLLO DE UN SISTEMA EN ROBODK UTILIZANDO MACHINE LEARNING PARA FACILITAR LA PROGRAMACIÓN DE PICK AND PLACE MEDIANTE BRAZOS MANIPULADORES DOBOT MAGICIAN EN SISTEMAS INDUSTRIALES.”

### **PROYECTO INTEGRADOR**

Previo a la obtención del Título de:

### **INGENIERO EN ELECTRÓNICA Y AUTOMATIZACIÓN INDUSTRIAL**

Presentado por:

Mannix Enmanuel Muñoz Villamar

GUAYAQUIL - ECUADOR

Año: 2022

## DEDICATORIA

A mis padres, por su amor y apoyo incondicional en todas las etapas de mi vida. Gracias por su paciencia, por enseñarme a nunca rendirme y por creer en mí siempre. Esta tesis es el resultado de sus enseñanzas y su amor, y es para ustedes con todo mi amor.

A mi tutor, por su orientación sabia y valiosa, por su paciencia y por su tiempo inestimable dedicado a ayudarme en este proyecto. Sus consejos y aliento han sido fundamentales para el éxito de esta tesis.

A novia, por su compañía y apoyo durante toda mi vida universitaria, por sus ánimos para que no me dé por vencido en el proceso de mi tesis. Sin toda su ayuda, esta tesis no habría sido posible.

## **AGRADECIMIENTOS**

A mi familia, por su constante amor y apoyo a lo largo de mi carrera académica. Sin su paciencia, comprensión y ánimo, no habría sido posible completar esta tesis.

A mi tutor de tesis, por su orientación experta, valiosos consejos y horas de revisión y discusión, Sin su dedicación y esfuerzo, no habría sido posible lograr esta tesis.

A todas las personas que de alguna forma contribuyeron a este trabajo, ya sea directa o indirectamente, Muchas gracias por su tiempo, ayuda y apoyo.

## DECLARACIÓN EXPRESA

“Los derechos de titularidad y explotación, me corresponde conforme al reglamento de propiedad intelectual de la institución; *Mannix Enmanuel Muñoz Villamar* y doy mi consentimiento para que la ESPOL realice la comunicación pública de la obra por cualquier medio con el fin de promover la consulta, difusión y uso público de la producción intelectual”



---

**Mannix Enmanuel Muñoz Villamar**  
**C.I. 0925850554**

## EVALUADORES



Escaneado y certificado por:  
DENNIS DICK CORTEZ  
ALVAREZ

---

**Ing. Dennys Cortez. MSc.**  
PROFESOR DE LA MATERIA

A handwritten signature in blue ink, appearing to read 'A. Prieto', located above the evaluator's name.

---

**Ing. Alexander Prieto. MSc.**  
PROFESOR TUTOR

## RESUMEN

Con el avance de la tecnología, empresas de distintos lugares tratan de automatizar sus sistemas de producción, despacho o empaquetamiento, para ello se ven en la necesidad de adquirir equipos de automatización, en donde destacan los robots industriales, frente a todo esto aparecen las dudas, ¿Cuál será la mejor forma de programar un robot industrial?, ¿Cuánto tiempo se demora este procedimiento?, ¿Qué pasará a futuro si se desea una nueva configuración?, entre otras preguntas. El objetivo es crear un sistema donde el robot conozca su entorno y pueda moverse según lo decida luego de ser entrenado con la finalidad que tome sus propias decisiones.

Con el fin de lograr que el robot sepa que movimientos realizar, se creó un sistema donde el robot Dobot Magician que se utilizará para probar el nuevo sistema, pueda mover sus articulaciones en su entorno, este entorno se creó en el software RoboDK y el entrenamiento es realizado utilizando la técnica de aprendizaje por reforzamiento específicamente el algoritmo de Q-Learning mediante el lenguaje de programación Python que dispone el mismo software para programar diferentes tipos de robots industriales.

Cuál fue el resultado de crear dicho sistema, se observa que se mejoró la precisión y el tiempo de programación para realizar la tarea de recoger un objeto y colocarlo en el lugar que se decide, según la necesidad del programador, adicional el código realizado puede ser utilizado en otros robots industriales, ya que el sistema se basa principalmente en el área de trabajo de los robots manipuladores que de alguna otra característica.

**Palabras Clave:** Dobot, Q-learning, Robot, RoboDK

## **ABSTRACT**

*With the advancement of technology, companies in different places try to automate their production systems, dispatch or packaging, for this they are in the need to acquire automation equipment, where industrial robots stand out, in the face of all this the doubts appear, What will be the best way to program an industrial robot, How long does this procedure take, What will happen in the future if you want a new configuration, among other questions. The goal is to create a system where the robot knows its environment and can move as it decides after being trained to make its own decisions.*

*In order to make the robot know what movements to make, a system was created where the Dobot Magician robot that will be used to test the new system, can move its joints in its environment, this environment was created in the RoboDK software and the training is done using the reinforcement learning technique specifically the Q-Learning algorithm using the Python programming language that has the same software to program different types of industrial robots.*

*What was the result of creating such a system, it is observed that the accuracy and programming time to perform the task of picking up an object and place it in the place that is decided, according to the need of the programmer was improved, additional the code made can be used in other industrial robots, since the system is mainly based on the work area of manipulator robots that of some other feature.*

*Keywords: Dobot, Q-learning, Robot, RoboDK.*

# ÍNDICE GENERAL

EVALUADORES .....	5
RESUMEN .....	I
<i>ABSTRACT</i> .....	II
ÍNDICE GENERAL.....	III
ABREVIATURAS .....	V
SIMBOLOGÍA .....	VI
ÍNDICE DE FIGURAS.....	VII
ÍNDICE DE TABLAS .....	IX
CAPÍTULO 1 .....	11
1. Introducción .....	11
1.1 Descripción del problema .....	11
1.2 Justificación del problema.....	12
1.3 Objetivos.....	13
1.3.1 Objetivo General .....	13
1.3.2 Objetivos Específicos .....	13
1.4 Marco teórico .....	14
1.4.1 Brazo robótico industrial.....	14
1.4.2 Tipos de brazos robóticos. ....	15
1.4.3 Dobot Magician .....	16
1.4.4 Limitaciones del Dobot Magician.....	18
1.4.5 Métodos De Programación para Robots Industriales. ....	19
1.4.6 Entorno de programación.....	20
1.4.7 Python en la programación de robots industriales.....	22
1.4.8 Machine learning.....	23



1.4.9	Q-learning	24
CAPÍTULO 2		26
2.	Metodología	26
2.1	Establecer requisitos y alcances	28
2.2	Elementos y datos de entrenamiento	29
2.3	Programación del código de entrenamiento	32
2.4	Entrenar y Validar	37
2.5	Implementar el sistema de programación del robot	38
CAPÍTULO 3		40
3.	Resultados Y ANÁLISIS	40
3.1	Aprendizaje supervisado	40
3.2	Aprendizaje por reforzamiento Q-Learning	41
3.3	Comparación entre los métodos utilizados	48
3.4	Comparación con una actividad real en la industria	49
CAPÍTULO 4		51
4.	Conclusiones Y Recomendaciones	51
	Conclusiones	51
	Recomendaciones	52
BIBLIOGRAFÍA		53
APÉNDICES		55

## **ABREVIATURAS**

ESPOL	Escuela Superior Politécnica del Litoral
ROS	Robotics Operating System
GUI	Interfaz gráfica de usuario
MDP	Markov Decision Process

## SIMBOLOGÍA

mm	Milímetro
cm	Centímetros

## ÍNDICE DE FIGURAS

Figura 1.1 Robot Industrial de Kuka (Robots, 2022) .....	15
Figura 1.2 Partes de un brazo robótico (INTEL, 2020) .....	16
Figura 1.3 Robot Dobot Magician (Dobot, s.f.).....	17
Figura 1.4 Articulaciones y espacio de trabajo del Robot Dobot Magician (Dobot, Dobot Magician User Manual, 2018).....	18
Figura 1.5 Programación y funcionamiento de un brazo robot.....	19
Figura 1.6 Interfaz Gráfica del Usuario GUI de RoboDK. (RoboDK, s.f.).....	22
Figura 2.1 Descripción del proceso que se realizó para obtener los resultados .....	27
Figura 2.2 Entorno creado, donde se muestra al robot y los objetos que se usarán para el entrenamiento. ....	28
Figura 2.3 Pseudocódigo del algoritmo Q-Learning. (Moya, 2022).....	30
Figura 2.4 Árbol de elementos y acciones RoboDK.....	33
Figura 2.5 Parte #1 código en Python.....	34
Figura 2.6 Parte #2 código en Python.....	34
Figura 2.7 Parte #3 código en Python.....	35
Figura 2.8 Parte #4 código en Python.....	35
Figura 2.9 Parte #5 código en Python.....	36
Figura 2.10 Espacio de trabajo del robot. ....	36
Figura 2.11 Ejemplo de entrenamiento y validación .....	37
Figura 2.12 Cuadro de diálogo para generar el programa del robot. ....	38
Figura 2.13 Código propio del robot exportado de roboDK.....	39
Figura 3.1 Botón de modo Teach en el robot Dobot Magician .....	40
Figura 3.2 Programación usando el modo Teach usando el robot Dobot Magician.....	41
Figura 3.3 Vista superior del espacio de trabajo del robot. ....	43
Figura 3.4 Vista frontal del espacio de trabajo del robot. ....	43
Figura 3.5 Vista lateral del espacio de trabajo del robot. ....	44
Figura 3.6 Posición inicial del robot .....	45
Figura 3.7 Resultado de simulación robot sujetando la caja .....	45
Figura 3.8 Resultado de la simulación, Robot trasladando la caja .....	46

Figura 3.9 Resultado de la simulación Robot dejando la caja sobre el pedestal. ...	46
Figura 3.10 Robot regresando a su posición de origen .....	46
Figura 3.11 Robot tomando la caja. ....	47
Figura 3.12 Robot moviéndose hasta el destino .....	47
Figura 3.13 Robot dejando el objeto en el destino.....	47
Figura 3.14 Proceso de empaquetado de Huevos usando un robot industrial.....	49

## ÍNDICE DE TABLAS

Tabla 2.1 Ejemplo de una Q-table que se obtuvo .....	31
Tabla 2.2 Comandos básicos de lenguaje Python para RoboDK. (python, s.f.) .....	32
Tabla 3.1 Límites negativos de las articulaciones del Dobot Magician .....	41
Tabla 3.2 3.3 Límites positivos de las articulaciones del Dobot Magician.....	42
Tabla 3.4 Posiciones de cada articulación .....	42
Tabla 3.5 Coordenadas de inicio y fin para el pick and place .....	44
Tabla 3.6 Comparación entre los métodos de programación estudiados. ....	48
Tabla 3.7 Comparación entre el método Teach y el aprendizaje por reforzamiento	49



# CAPÍTULO 1

## 1. INTRODUCCIÓN

Las primeras aplicaciones de robots manipuladores surgieron en la década de los años 60 en tareas orientadas al control numérico en la industria y a la movilización de cargas pesadas. Sin embargo, en ese momento la autonomía de los robots era nula y se basaba en movimientos predefinidos sin realimentación sensorial. Con el tiempo, el desarrollo de la tecnología permitió que los robots industriales adquirieran una mejor tecnología de sensores y una cierta capacidad de inteligencia para reaccionar a los cambios en su entorno. A partir de principios de la década de 2000, los avances en robótica industrial han sido impulsados en gran medida por los avances a nivel de software, especialmente en el campo del aprendizaje automático y la visión por computador (Rus, 2015). El aprendizaje por refuerzo es una rama importante del aprendizaje automático que se utiliza para establecer secuencias de decisiones óptimas para agentes con el objetivo de maximizar un cierto resultado esperado, y se ha aplicado en una gran variedad de problemas tanto virtuales como reales, incluyendo tareas de control continuo en la manipulación de objetos mediante dispositivos robóticos.

### 1.1 Descripción del problema

En Ecuador el uso de brazos robots para realizar tareas industriales ha ido aumentando, por tanto, para adquirir un equipo, siempre se debe pensar también en su configuración y programación. Para esto se necesita tiempo considerable para analizar el área de trabajo, la tarea a realizar, que objetos va a manipular y como los va a manipular, para finalmente realizar una programación y configuración precisa y efectiva. A consecuencia de esto, si algo no sale bien o se desea cambiar ciertas posiciones a la que debe llegar el robot, ya sea para dejar o recoger un objeto se debe realizar un cambio en la programación y en la rutina de este para las nuevas tareas, esta acción



podría tornarse un poco difícil y costosa si se requiere cambiar repetidas veces las rutinas del robot.

El uso de la técnica Machine Learning nos permite crear un sistema para lograr una solución muy efectiva a esta problemática, ya que da al robot la posibilidad de tomar sus propias decisiones según los ejemplos que se le haya dado en el entrenamiento para realizar una tarea determinada de forma más rápida y eficaz sin necesidad de una nueva programación o cambio en sus rutinas.

Técnicas de programación como, modo teach, modo paso a paso u otras, a pesar de su sencillas, requieren la instalación o montaje del robot en el espacio de trabajo y de ejecuciones reales, que pueden ser mucho más complejas si las dimensiones del robot son grandes, por lo que se requiere un sistema que permita la programación sin estos requerimientos que en la actualidad no existe.

## **1.2 Justificación del problema**

La realización de un entrenamiento para la programación de robots industriales brinda diferentes ventajas, reduce el tiempo y el costo de programación y configuración de dicho robot.

Para implementar esta técnica de aprendizaje se usará una herramienta de entrenamiento como es el Q-learning (**aprendizaje por reforzamiento**), la cual consiste en entrenar al robot para que aprenda una serie de reglas las cuales usará en el futuro para realizar una determinada acción en una determinada circunstancia tomando sus propias decisiones, algo que no es posible usando las técnicas tradicionales de programación para brazos manipuladores.

Todo este entrenamiento se lo realizará en un entorno de simulación. El software ROBODK brinda la posibilidad de simular casi todos los robots industriales fabricados hasta la fecha, en donde se puede diseñar y entrenar el robot en un entorno familiar y para dicho entrenamiento se usará la técnica de Q-Learning utilizando el lenguaje de programación PYTHON, para finalmente ser implementado utilizando el robot DOBOT MAGICIAN del laboratorio de robótica industrial de ESPOL.

### **1.3 Objetivos**

#### **1.3.1 Objetivo General**

Desarrollar un sistema en RoboDK mediante entrenamiento utilizando aprendizaje por reforzamiento (Q-Learning) para facilitar la programación del robot Dobot Magician para tareas de pick and place.

#### **1.3.2 Objetivos Específicos**

1. Comparar la técnica de Machine Learning Q-Learning con otras técnicas de aprendizaje para demostrar que es la más idónea.
2. Definir las características principales del robot Dobot Magician y su entorno para su programación.
3. Diseñar un entorno familiar en el software RoboDK para el entrenamiento del sistema a desarrollar.
4. Diseñar el código de entrenamiento basado en el algoritmo del modelo Q-Learning en lenguaje Python para el desarrollo del sistema de programación del robot.
5. Obtener el sistema de programación automático simulado mediante el entrenamiento diseñado.

## 1.4 Marco teórico

### 1.4.1 Brazo robótico industrial

Los brazos robóticos industriales son uno de los tipos más comunes de robots en uso hoy en día. Son especialmente útiles para realizar tareas repetitivas y precisas en una variedad de entornos, como la fabricación, el sector automotriz y la agricultura. Los brazos robóticos articulados pueden programarse para realizar una amplia gama de tareas y se están utilizando cada vez más en aplicaciones industriales para aumentar la eficiencia y reducir los costos laborales. Además, las nuevas tecnologías y conectividad están permitiendo que los brazos robóticos se utilicen en nuevos casos prácticos y modelos de negocio. Los brazos robóticos industriales ofrecen varios beneficios para las empresas (Robots, 2022). Algunos de los principales beneficios incluyen:

- **Seguridad mejorada:** Los brazos robóticos pueden operar en entornos peligrosos y realizar tareas que representan un gran riesgo de lesión para los humanos.
- **Eficiencia y productividad mejoradas:** Los brazos robóticos pueden operar sin descanso las 24 horas del día, los siete días de la semana. Esto permite a las empresas mantener la producción y realizar inspecciones u otras tareas de manera continua para aumentar los resultados.
- **Precisión mejorada:** Debido a su naturaleza, los brazos robóticos funcionan de manera más consistente y precisa que los humanos en tareas que requieren una precisión extrema.
- **Mayor flexibilidad:** los brazos robóticos pueden adaptarse a una variedad de tareas y entornos, y se pueden reutilizar para nuevas actividades a medida que cambian las prioridades empresariales. También pueden montarse en diferentes plataformas, como robots

móviles autónomos (AMRs), una plataforma de línea de ensamblaje fija, en la pared o en un estante. (Robots, 2022)



**Figura 1.1 Robot Industrial de Kuka (Robots, 2022)**

#### **1.4.2 Tipos de brazos robóticos.**

Existen varios tipos de robots manipuladores, cada uno con características y capacidades específicas.

##### **El robot cartesiano**

Se caracteriza por sus ejes que coinciden con los tres ejes cartesianos y se utiliza en funciones de soldadura, operaciones de ensamblado y manipulación de objetos.

##### **El robot esférico o polar**

Tiene ejes que forman un sistema polar de coordenadas y se utiliza para tareas de soldadura, fundición y manipulación de máquinas y herramientas.

##### **El robot articulado**

Tiene al menos tres articulaciones rotatorias, lo que le permite llevar a cabo tareas más complejas, como la soldadura y la pintura con aerosol.

##### **El robot cilíndrico**

Tiene ejes que forman un sistema de coordenadas de círculos concéntricos y se utiliza para tareas de manipulación de máquinas y soldadura de punto.

## El robot SCARA

Tiene dos articulaciones rotatorias paralelas y se utiliza para tareas de pick and place, como la recolección y colocación de objetos.

## El robot paralelo

Tiene brazos con articulaciones en forma de prisma y se utiliza principalmente en plataformas móviles para simulaciones de vuelo. (School, 2019)

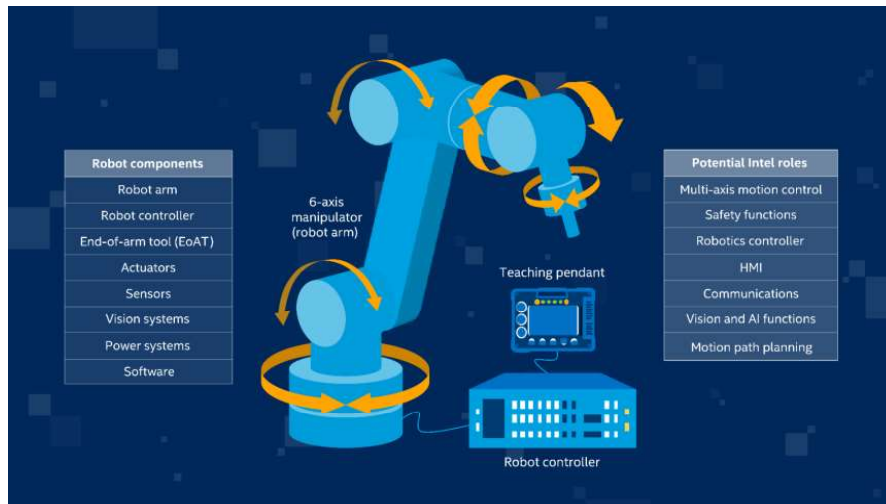


Figura 1.2 Partes de un brazo robótico (INTEL, 2020)

### 1.4.3 Dobot Magician

Es un brazo robótico industrial de escritorio con un alto nivel de precisión y versatilidad, desarrollado por la empresa china Shenzhen Yuejiang Technology Co. Ltd, es un brazo robótico que puede moverse hacia arriba, hacia abajo, hacia adelante o hacia atrás, hacia la derecha o hacia la izquierda, y girar en el plano horizontal.

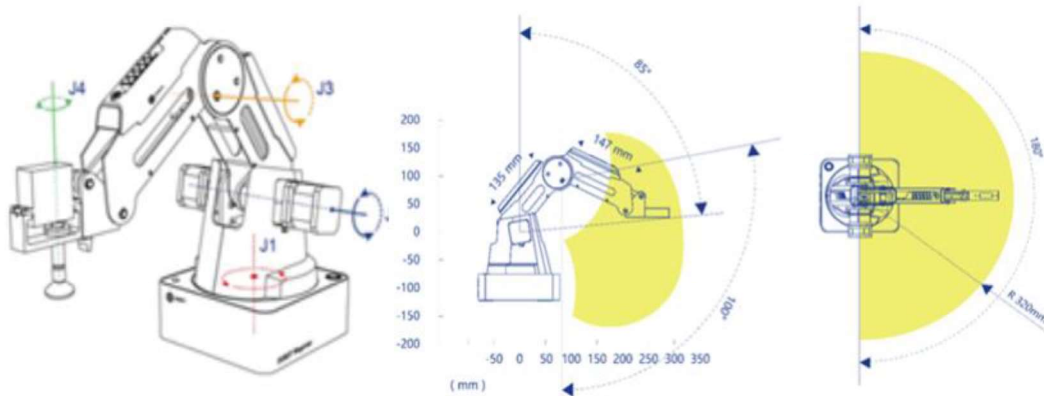
Dobot Magician se caracteriza por sus cuatro ejes de movimiento y por su precisión de 0,2 mm. Es capaz de realizar tareas de soldadura, escritura, dibujo, ensamblaje, entre otras. El brazo robótico cuenta con un software específico que permite la programación en diferentes lenguajes, como Python, C++, C#, LabVIEW, entre otros. (Dobot, Dobot, s.f.)

Dobot Magician se utiliza comúnmente en aplicaciones educativas, para enseñanza de programación y automatización, pero también puede ser utilizado en industrias como la electrónica, la medicina y la fabricación. Además, cuenta con una gran cantidad de accesorios y herramientas intercambiables que amplían las posibilidades de uso del robot, desde sujetadores para diferentes tipos de objetos hasta herramientas de soldadura, escritura, dibujo, entre otros.



**Figura 1.3 Robot Dobot Magician (Dobot, s.f.)**

#### 1.4.4 Limitaciones del Dobot Magician.



**Figura 1.4 Articulaciones y espacio de trabajo del Robot Dobot Magician (Dobot, Dobot Magician User Manual, 2018)**

Aunque el Dobot Magician es un brazo robótico industrial de escritorio con alta precisión y versatilidad, todavía tiene algunas limitaciones. Algunas de las limitaciones más comunes del Dobot Magician incluyen:

**Capacidad de carga:** El Dobot Magician tiene una capacidad de carga limitada debido a su tamaño compacto. Por lo general, no es adecuado para tareas que requieren levantar objetos pesados.

**Rango de movimiento:** El Dobot Magician tiene un rango de movimiento limitado en comparación con los brazos robóticos industriales de mayor tamaño. Aunque cuenta con 4 grados de libertad, el rango de movimiento es limitado y no cubre tareas que requieren un alcance mayor.

**Velocidad de movimiento:** Aunque el Dobot Magician es capaz de realizar movimientos precisos, la velocidad de movimiento no es tan rápida como la de algunos brazos robóticos industriales más grandes. Esto puede afectar la eficiencia de la línea de producción en aplicaciones industriales.

#### 1.4.5 Métodos De Programación para Robots Industriales.

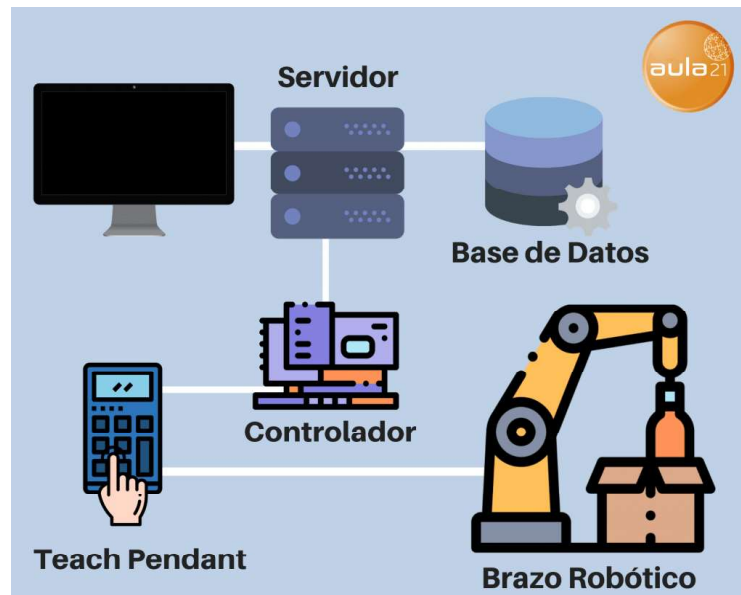


Figura 1.5 Programación y funcionamiento de un brazo robot

Existen varios métodos de programación para brazos robóticos, cada uno con sus propias ventajas y desventajas. Algunos de los métodos de programación más comunes incluyen:

- **Programación por ensayo y error:** Este método implica mover manualmente los ejes del brazo robótico y registrar las posiciones finales para luego programar las mismas secuencias de movimiento. Este método es fácil de usar, pero puede llevar mucho tiempo para programar una tarea compleja.
- **Programación por lenguaje de comando:** Este método implica escribir órdenes de programación en un lenguaje de programación específico para controlar los movimientos del brazo robótico. Este método es más preciso y permite la programación automatizada de tareas complejas, pero requiere conocimientos de programación.



- **Programación por enseñanza:** Este método implica mover manualmente los ejes del brazo robótico y registrar las posiciones finales para luego reproducir las mismas secuencias de movimiento. Este método es fácil de usar y permite una programación rápida, pero puede ser menos preciso que otros métodos.
- **Programación por visualización:** Es un método en el que se utiliza un software específico para visualizar el brazo robótico y su entorno, y luego se programa la tarea utilizando una interfaz gráfica. Es fácil de usar y permite una programación rápida y precisa. (industrial, 2018)

#### 1.4.6 Entorno de programación.

El entorno de programación de robots industriales se refiere a la plataforma o conjunto de herramientas utilizadas para programar y controlar el comportamiento de un brazo robótico industrial. Los entornos de programación de robots industriales pueden variar dependiendo del fabricante del robot y del sistema de control utilizado.

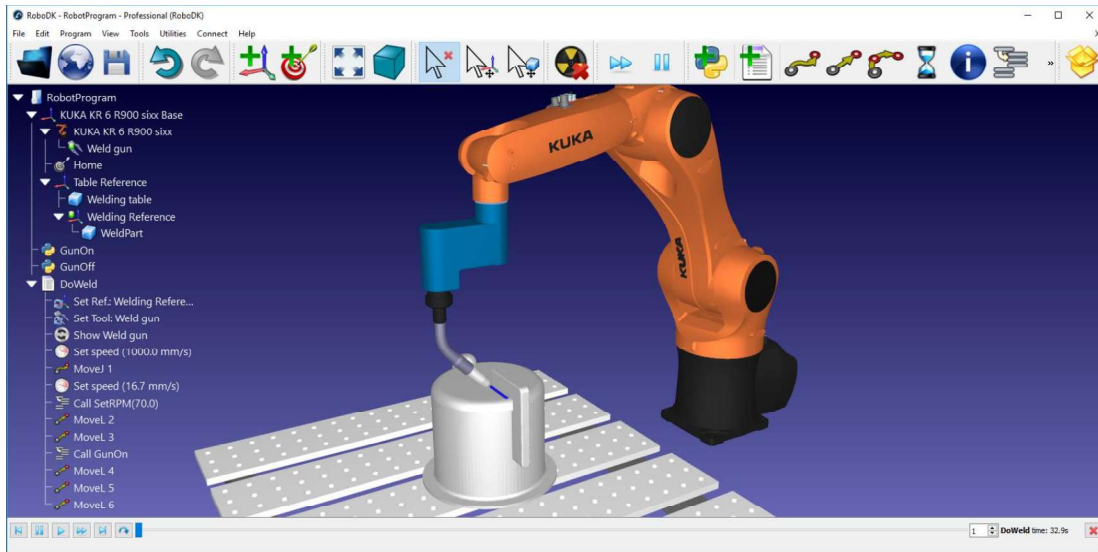
Algunos fabricantes proporcionan software específico para programar y controlar sus robots, mientras que otros utilizan entornos de programación estándar, como lenguajes de programación de alto nivel, como Python, C++, C#.

Cada entorno tiene sus propias ventajas y desventajas, y es importante elegir el entorno de programación adecuado para el robot y la aplicación específica, algunos ejemplos de entornos de programación de robots industriales incluyen:

- **Robotics Operating System (ROS):** Es una plataforma de código abierto para programar robots, permite la comunicación entre los diferentes componentes del sistema, tales como sensores, actuadores y computadoras.

- **Universal Robots PolyScope:** Es un sistema de programación basado en GUI (interfaz gráfica de usuario) para programar robots de la marca Universal Robots
- **Fanuc Robotics:** Ofrece una variedad de software para programar y controlar sus robots, incluyendo su sistema de control robótico R-30iA.
- **Kuka Robotics:** Proporciona herramientas de programación, tales como el software Kuka.Sim Pro, que permite programar robots de la marca Kuka mediante un entorno de simulación. (industrial, 2018)
- **RoboDK:** RoboDK es un software de programación y simulación para robots industriales. Permite a los usuarios programar y simular robots de diferentes fabricantes, incluyendo brazos robóticos como ABB, Fanuc, KUKA, Motoman, UR, entre otros. Este software permite crear programas de robótica en un ambiente virtual con una interfaz gráfica de usuario (GUI) amigable, facilitando la programación, la simulación y la generación de código para los robots. (RoboDK, s.f.)  
RoboDK cuenta con una gran cantidad de herramientas para programar robots, como:

- ✓ Interfaz gráfica de usuario (GUI) para programar robots mediante movimientos en 3D.
- ✓ Simulación para validar y probar programas antes de implementarlos en robots físicos.
- ✓ Generación de código para diferentes lenguajes de programación, incluyendo Python, C++, C#, Matlab, entre otros.
- ✓ Interfaz para conectar con sensores y dispositivos de adquisición de datos.
- ✓ Compatibilidad con diferentes sistemas operativos, incluyendo Windows, Linux y Mac OS. (RoboDK, s.f.)



**Figura 1.6 Interfaz Gráfica del Usuario GUI de RoboDK. (RoboDK, s.f.)**

### **1.4.7 Python en la programación de robots industriales.**

Python es un lenguaje de programación ampliamente utilizado en la robótica y la automatización industrial debido a su facilidad de uso, su gran cantidad de bibliotecas y su alto desempeño. Muchas bibliotecas de robótica en Python se han desarrollado para facilitar el control de robots, la simulación y el procesamiento de datos.

Algunas de las ventajas de usar Python para programar robots industriales incluyen:

- Fácil acceso a bibliotecas de robótica y automatización industrial que facilitan la programación de robots.
- Mayor productividad y rapidez en la programación debido a su sintaxis intuitiva y fácil de usar.
- La capacidad de programar robots en tiempo real y procesar datos de sensores en tiempo real.
- Comunidad de desarrollo grande y activa, lo que significa que hay muchos recursos y bibliotecas disponibles en línea. (Heras, 2020)

Algunos ejemplos de bibliotecas de Python utilizadas para programar robots industriales incluyen:

- ✚ **ROSPy:** Una implementación de ROS (Robotics Operating System) en Python, que permite el acceso a las herramientas de control de robots de ROS.
- ✚ **PyRobot:** Una biblioteca de Python para controlar robots y hacer simulaciones de robot.
- ✚ **Pycontrol:** Una biblioteca de Python para controlar robots de plataformas como ABB, KUKA, UR, entre otros.
- ✚ **Scipy, numpy:** Bibliotecas de cálculo científico y numérico que ayudan en la toma de decisiones y en el procesamiento de datos del robot. (Heras, 2020)

#### 1.4.8 Machine learning.

Machine learning es una rama de la inteligencia artificial que se enfoca en el desarrollo de algoritmos y técnicas que permiten a las máquinas aprender a partir de datos, sin ser explícitamente programadas. El objetivo principal del aprendizaje automático es desarrollar sistemas que puedan mejorar su rendimiento a medida que adquieren más datos.

Existen varios tipos de algoritmos de aprendizaje automático, que se pueden clasificar en tres categorías principales:

- **Aprendizaje supervisado:** Se utiliza para aprender a clasificar o predecir una variable objetivo a partir de un conjunto de datos etiquetados. Ejemplos comunes incluyen algoritmos de clasificación como regresión logística o redes neuronales.
- **Aprendizaje no supervisado:** Se utiliza para aprender patrones y relaciones en datos no etiquetados. Ejemplos comunes incluyen algoritmos de agrupamiento como el algoritmo k-means o el análisis de componentes principales. (IBM, s.f.)

- **Aprendizaje por refuerzo:** Es una técnica de aprendizaje automático en la cual un agente aprende a maximizar una recompensa a través de la interacción con el entorno. El agente toma acciones y recibe recompensas o castigos, utilizando esta información para mejorar sus acciones en el futuro.

El aprendizaje por refuerzo se basa en la teoría del refuerzo, que sostiene que la probabilidad de que un comportamiento se repita aumenta si es seguido de una recompensa, y disminuye si es seguido de un castigo. El objetivo del agente en el aprendizaje por refuerzo es encontrar la mejor política de acción, es decir, el conjunto de acciones que maximizan la recompensa a largo plazo.

Este tipo de aprendizaje se utiliza en una gran variedad de aplicaciones, como la toma de decisiones en sistemas de control automático, la optimización de procesos industriales, el diseño de videojuegos, la inteligencia artificial en robótica, entre otros, es importante mencionar que el aprendizaje por refuerzo no requiere de grandes cantidades de datos. (TEAM, 2020)

#### **1.4.9 Q-learning.**

Q-Learning es un algoritmo de aprendizaje por refuerzo que se utiliza para aprender una política de acción óptima en un ambiente de Markov Decision Process (MDP). El objetivo del algoritmo es aprender una función Q que asigne un valor a cada estado-acción del sistema, donde el valor de un estado-acción representa la recompensa esperada a largo plazo si se toma esa acción en ese estado.

El algoritmo Q-Learning se basa en el principio de la búsqueda de la acción óptima. El agente explora el entorno tomando acciones y actualizando su valor Q a medida que recibe recompensas o castigos. La acción

seleccionada en cada estado es la que tiene el mayor valor Q, y el agente actualiza su valor Q a medida que adquiere nueva información sobre el entorno.

El algoritmo Q-Learning es uno de los métodos de aprendizaje por refuerzo más utilizado, ya que es fácil de implementar y es escalable a problemas complejos. Sin embargo, tiene algunas limitaciones, como la necesidad de almacenar y actualizar una tabla Q para cada estado posible, lo que puede ser computacionalmente costoso en problemas con un gran número de estados, a pesar de ello, Q-learning se ha aplicado con éxito en una gran variedad de aplicaciones, como la optimización de procesos industriales, la conducción autónoma, entre otras. (TEAM, 2020)

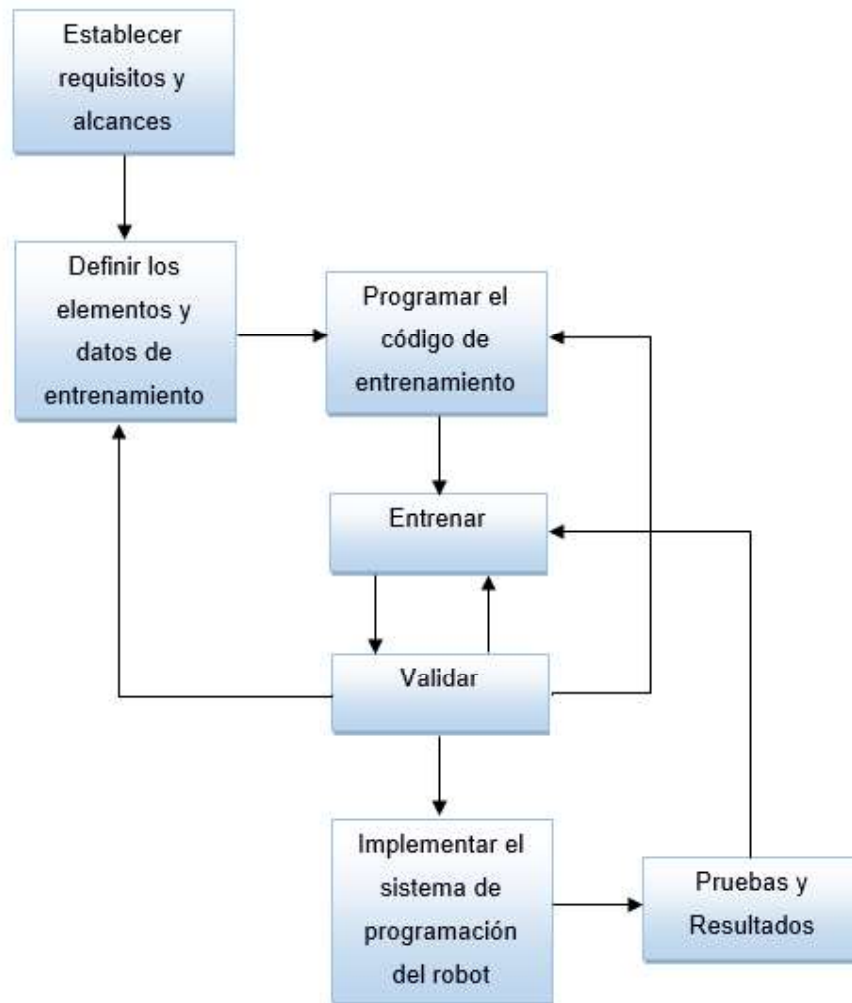
# CAPÍTULO 2

## 2. METODOLOGÍA

El algoritmo Q-Learning se puede utilizar en la programación de brazos robóticos de manera similar a como se utiliza en otros problemas de aprendizaje por refuerzo. El objetivo sería aprender una política de acción óptima para el brazo robótico a través de la interacción con el entorno.

Para implementar Q-learning en un brazo robótico, se pueden seguir los siguientes pasos:

1. Definir el estado del sistema: el estado del sistema puede incluir información sobre la posición y orientación del brazo robótico, la posición de los objetos en el entorno, entre otros.
2. Definir las acciones disponibles: las acciones disponibles pueden incluir movimientos específicos del brazo robótico, como el movimiento de los ejes o la apertura y cierre de las pinzas.
3. Implementar la función Q: se puede implementar una tabla Q para almacenar los valores Q para cada estado-acción.
4. Utilizar el algoritmo Q-Learning para actualizar la tabla Q: el agente debe tomar acciones y actualizar su valor Q a medida que recibe recompensas o castigos.
5. Utilizar la política óptima para controlar el brazo robótico: una vez que se ha aprendido una política óptima, se puede utilizar la función Q para controlar el brazo robótico en tiempo real.



**Figura 2.1 Descripción del proceso que se realizó para obtener los resultados**

Para poder obtener el sistema de entrenamiento y validarlo con el robot, se plantea seguir los pasos mencionados en el diagrama de la figura 2.1:

Primero cumpliremos los requerimientos y dejaremos definido todos los equipos con los que se trabajará, luego crear el entorno en el cual el robot va a familiarizarse.

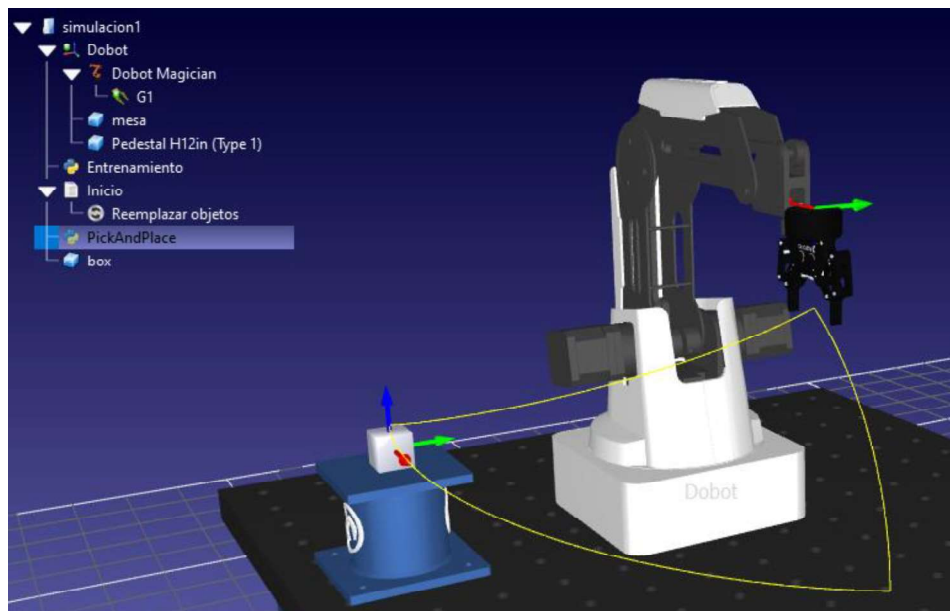
Seguido de esto la creación del código de entrenamiento utilizando Python para luego ir validando con el robot, con el fin de observar si el robot puede alcanzar todos los puntos que se obtienen en el entrenamiento.



Finalmente, si se desea cargar este programa al robot, roboDK nos permite exportar un archivo que nos muestra un código parecido a la programación que el robot usa.

## 2.1 Establecer requisitos y alcances.

Para realizar el entrenamiento del brazo robot se utilizó un entorno desarrollado en el programa RoboDK, software de simulación y programación de robots industriales. Permite a los usuarios diseñar, simular y generar programas para robots de varias marcas. También ofrece una interfaz gráfica fácil de usar para la creación de programas y una biblioteca de herramientas y procesos para automatizar tareas específicas. Además, permite importar modelos 3D de piezas y equipos para simular un entorno de producción completo. (RoboDK, s.f.)



**Figura 2.2 Entorno creado, donde se muestra al robot y los objetos que se usarán para el entrenamiento.**

Como se puede observar, se usó un robot Dobot Magician para la programación y prueba en el cual realizó un pick and place moviendo la caja con dimensiones de 2.5 cm por lado, el tamaño del pedestal puede variar o

incluso se puede utilizar la base de la mesa, el punto de origen es el lugar donde se coloca la caja y el final es el punto que se seleccione dentro de los límites de alcance del robot.

## 2.2 Elementos y datos de entrenamiento.

Q-learning es un algoritmo de aprendizaje por refuerzo que utiliza una tabla Q para almacenar la valoración de cada acción en un estado determinado. El objetivo de Q-learning es encontrar la política óptima que maximiza la recompensa a largo plazo, donde el agente toma acciones basadas en la información almacenada en su tabla Q. Q-learning es un algoritmo de aprendizaje por ensayo y error, y a medida que el agente experimenta más, su tabla Q se actualiza y su estrategia se mejora. (Moya, 2022)

Para realizar lo antes mencionado se hace utilizó la “**Q-funtion**” definida como:

$$\widehat{Q}(s, a) = Q(s, a) + \alpha * [R(s, a) + (\gamma * \max_{a'} Q(s', a')) - Q(s, a)]$$

Donde:

- $\widehat{Q}(s, a)$ : Es el nuevo valor de  $Q(s, a)$
- $Q(s, a)$ : Es el valor actual de  $Q(s, a)$
- $\alpha$ : Nos indica cuanto queremos aprender en cada acción.
- $R(s, a)$ : Recompensa por tomar la acción “a” luego del estado actual “s”.
- $\gamma$ : Factor de descuento
- $\max_{a'} Q(s', a')$ : Es el valor de  $Q(s, a)$  máximo para tomar la mejor acción posible. (Moya, 2022)

Esta función al implementada en Python nos ayuda a crear una tabla con valores de Q para cada acción que el robot realiza, el pseudocódigo de este algoritmo se presenta de la siguiente forma:

```
Q(,) = 0 # Inicialización Política
for each episodio do
    s = estado inicial
    for each paso, estado s (no final) do
         $a = \begin{cases} \text{random}(\text{accion}) & \rightarrow \text{random}() \leq \text{ratio\_exploración} \\ \max_a Q(s, \cdot) & \rightarrow \text{random}() > \text{ratio\_exploración} \end{cases}$ 
        realizar acción a
        s = s' # Nuevo estado
        R(s,a) = reward
         $\hat{Q}(s, a) = Q(s, a) + \alpha [R(s, a) + (\gamma \cdot \max_{a'} Q(s', a')) - Q(s, a)]$ 
    end
end
```

Figura 2.3 Pseudocódigo del algoritmo Q-Learning. (Moya, 2022)

Una breve descripción pseudocódigo:

- El proceso se repetirá un número determinado de veces, esto lo tomaremos como la cantidad de pasos que queremos entre el origen y el destino.
- Por cada iteración el agente siempre partirá del estado inicial por tanto se tomarán varios valores de Q durante la iteración.
- Dentro de cada iteración los valores de las acciones que puede realizar el robot se irán guardando en la tabla con el fin que luego él tome la decisión de cual acción es mejor y seguir el camino hasta su destino.

- Por cada estado el agente puede realizar las acciones de dos formas distintas, la primera sería explorando, que significa que realiza una acción al azar y la otra forma es explotando, que significa tomar la mejor acción entre las disponibles.
- Todo esto se repite hasta tener el mejor camino y el robot llegue a su destino.

La Q-Table ayuda al agente a tomar decisiones óptimas en base a la experiencia previa y su conocimiento acumulado. (Moya, 2022)

- La Q-Table puede ser representada como una tabla donde las filas corresponden a los estados y las columnas corresponden a las acciones.
- Cada celda de la tabla contiene el valor Q asociado con una acción en un estado específico.

Los valores de la **Q-table** se van actualizando según la función “**Q-funtion**”.

<b>Estado</b>	<b>Acción 1</b>	<b>Acción 2</b>	<b>Acción 3</b>	<b>Acción 4</b>
<b>S1</b>	0,0	0,5	0,0	0,6
<b>S2</b>	0,0	2,3	0,1	4,7
<b>S3</b>	0,1	6,4	0,0	8,9
...	...	...	...	...
<b>Sn</b>	3,6	0,0	7,2	0,0

**Tabla 2.1 Ejemplo de una Q-table que se obtuvo.**

### 2.3 Programación del código de entrenamiento.

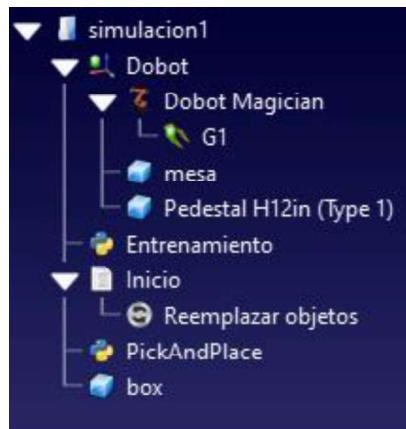
La API de RoboDK es una interfaz de programación que permite la integración de RoboDK con un lenguaje de programación externo, lo que facilita la automatización de tareas y la programación fuera de línea de robots industriales.

La API de RoboDK permite acceder a funcionalidades de RoboDK, como la creación y control de objetos en la simulación, la generación de programas para controladores de robots específicos, y la automatización de tareas repetitivas dentro del simulador. (python, s.f.)

Para poder utilizar esta interfaz se usaron los comandos básicos que utiliza RoboDK para la programación en Python, algunos de ellos se muestran a continuación.

Comando	Significado	Ejemplo
<i>Item</i>	Permite seleccionar de forma directa el objeto o robots, target, etc. Para utilizarlo en la programación Python	Item('box100mm') reference = Item ('Kawasaki RS03N Base') robot = Item ('Kawasaki RS03N')
<i>Pose ()</i>	Devuelve la posición relativa de un objeto o marco de referencia con respecto a su padre.	poserobot = robot. Pose ()
<i>JointsHome</i>	Devuelve los valores de los ángulos de las articulaciones del robot para la posición Home.	robot. JointsHome ()
<i>MoveJ,</i> <i>MoveL</i>	Permite el movimiento del robot de forma Joint o lineal.	robot. MoveJ (transl [850,300,10])
<i>JointLimits ()</i>	Devuelve los límites de las articulaciones del robot previamente seleccionado.	robot. JointLimits ()
<i>Collision ()</i>	Detecta colisión entre 2 objetos, devuelve 1 si están en colisión.	Objeto1.Collision(objeto2)
<i>setPose ()</i>	Permite modificar la ubicación u orientación de un objeto en la estación	Objeto1.setPose([250,350,10])

**Tabla 2.2 Comandos básicos de lenguaje Python para RoboDK. (python, s.f.)**



**Figura 2.4** Árbol de elementos y acciones RoboDK

En la figura 2.4 se observa los elementos necesarios para la programación del robot que se describen como:

- ❖ **Dobot Magician** : Es nuestro robot a utilizar.
- ❖ **G1**: Nombre de la pinza de agarre, herramienta del robot
- ❖ **Mesa**: Es nuestra base, donde se encontraran todos los elementos.
- ❖ **Pedestal H12in**: Es la superficie donde se colocará la caja en la ejecución del programa, sería nuestro destino en el pick and place
- ❖ **Entrenamiento**: Archivo de programación con el código de entrenamiento
- ❖ **Reemplazar objetos**: Para fines de simulación se coloca una acción que nos permitirá volver todo a su estado original cada vez que se ejecute la simulación.
- ❖ **PickAndPlace**: Contiene el código para el movimiento de tomar el objeto y llevarlo a su destino final.
- ❖ **Box**: Sera nuestro objeto, una caja de 2 cm por lado.

En el archivo con nombre “Entrenamiento” tenemos el siguiente código:

```
from robodk.robolink import * # RoboDK API
from robodk.robomath import *
from robodk import *
from time import *
from numpy import *
```

**Figura 2.5 Parte #1 código en Python**

Para esta parte tenemos las librerías que usa RoboDK en python, estas ayudarán en la simulación dando acceso al uso de las funciones propias del simulador, además de la librería “time” que nos permite colocar temporizadores y la librería “numpy” que nos permite realizar tareas usando las matrices y vectores de una forma más sencilla.

```
RDk = Robolink()
robot = RDk.ItemUserPick('Dobot Magician', ITEM_TYPE_ROBOT)
G1 = RDk.Item('G1', ITEM_TYPE_TOOL)
RDk.RunProgram('Inicio')

if not robot.Valid():
    quit()
```

**Figura 2.6 Parte #2 código en Python**

En esta sección definimos los nombres de los objetos con los que se trabajará dentro de la programación, como son el robot, G1, y además ejecutamos el programa “Inicio” este programa devolverá todos los objetos a su estado inicial cada vez que se ejecute el programa.

Adicionalmente colocaremos una en la línea 6 una condición que nos permite ejecutar el programa siempre y cuando exista un robot definido en el sistema.

```

home_joints = robot.JointsHome()
robot.MoveJ(home_joints)
limitneg,limitpos,pp = robot.JointLimits()
print ("Limites: Positivos\n",limitneg, "\nNegativos\n",limitpos ,"\n" )

```

**Figura 2.7 Parte #3 código en Python**

En esta sección, en la primera línea se obtiene la posición de Home del robot, es decir la posición original que toma en toda simulación antes de ejecutarse algún código. En la segunda línea se procede a mover al robot a esa ubicación.

En la línea 3 obtendremos los límites del robot, es decir las posiciones máximas que puede alcanzar cada articulación del robot.

```

def matrizPos(limiPos,limiNeg):
    listaneg=list(limiNeg)[0] #lista de limites negativos
    listaPos=list(limiPos)[0] # lista de limites positivos
    matriz=[] # matriz donde se guardan las posiciones

    numero=4 # numero de pasos entre inicio y fin
    for i in range(len(listaneg)):

        # se obtiene los valores que debe tomar cada
        # articulación según se vaya moviendo
        paso=abs(listaneg[i]-listaPos[i])/numero
        posArti=[]
        for j in range(numero):
            posArti.append(listaneg[i]+(paso*j))
        posArti.append(listaPos[i])
        matriz.append(posArti) # guardamos los datos
    return array(matriz)

matrizPosi=matrizPos(limitpos,limitneg)
print(matrizPosi)

```

**Figura 2.8 Parte #4 código en Python**

La función descrita en la figura 2.8 nos ayudará a crear la matriz de posiciones de tal forma que tomará los límites positivos y negativos del robot y creará la matriz con todas las posiciones que el robot puede



alcanzar, la cantidad de pasos entre los límites positivos y negativos los dará la variable “numero” descrita en el código en la línea 6.

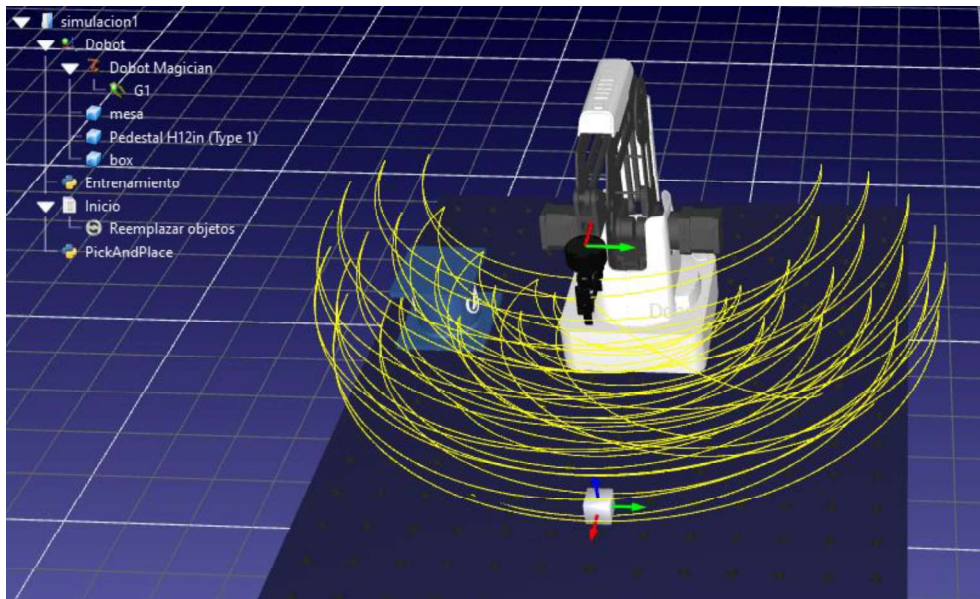
```
matrizPosi=matrizPos(limitpos,limitneg)
print(matrizPosi)

def mostrarMovi():
    for j in matrizPosi[2,:]:
        for k in matrizPosi[1,:]:
            for l in matrizPosi[0,:]:
                robot.MoveJ([l,k,j,0])
    return ""

mostrarMovi()
robot.MoveJ(home_joints)
```

**Figura 2.9 Parte #5 código en Python**

Como parte final del archivo tenemos la función “mostrarMovi()”, que nos permitirá recorrer la matriz de posiciones y nos ayudará a simular los movimientos dentro del área de trabajo del robot, es decir las posiciones a las que el robot alcanza, tal como se muestra a continuación.



**Figura 2.10 Espacio de trabajo del robot.**

En la figura 2.10 se observa de líneas amarillas, serían los caminos que el robot toma al moverse en su espacio de trabajo entre los límites de sus articulaciones.

## 2.4 Entrenar y Validar

Como el robot ya conoce su lugar de trabajo, se procede a indicarle en que posición se debe recoger la caja y hasta que lugar debe llevarla, en la siguiente figura se muestra un ejemplo de como se valida que el robot realice la tarea de pick and place.

```
pick=[0.11,64.10,39.35,84.11]
place=[-89.98,19.81,36.28,-5.98]

# caja
robot.MoveJ(pick)
TCP_On(G1)
robot.MoveJ(place)
TCP_Off(G1)
robot.MoveJ(home_joints)
```

**Figura 2.11 Ejemplo de entrenamiento y validación**

Describiendo la figura 2.11 tenemos. En las dos primeras líneas, le damos al robot los puntos de inicio (pick) y el final (place), esto quiere decir que el robot se moverá entre estos dos puntos según el camino que haya decidido, este camino es seleccionado según los coeficientes de Q que le indican cuales son los puntos mas eficientes por los cuales se debe mover el robot sin chocar con algun obstáculo.

La función “moveJ()” según su descripción en la tabla 2.2 nos permite mover al robot hasta el punto colocado como parámetro. La función TCP\_On() y TCP\_Off() nos ayuda a activar y desactivar las pinzas que tiene como herramienta el robot. Luego que termina de recorrer su camino y dejar el objeto, en la última línea le indicamos al robot que se mueva hasta su posición original.

Si el robot realiza su tarea satisfactoriamente, se podría proceder a guardar los puntos que usa como camino, ya que serían los mas eficientes y por tanto ya no se necesita volver a entrenar al robot en su entorno. Para visualizar los movimiento que el robot realizó, se pueden observar desde la figura 3.6 hasta la figura 3.10 del respectivo capítulo 3.

## 2.5 Implementar el sistema de programación del robot.

Con los puntos de inicio y final definidos, además de tener el camino que el robot va a seguir hasta completar su tarea, se procedió a generar el código propio en el lenguaje que el robot entienda, este código es propio de cada robot industrial y se genera dando clic derecho sobre el archivo que deseamos convertir, en este caso el archivo "PickAndPlace".

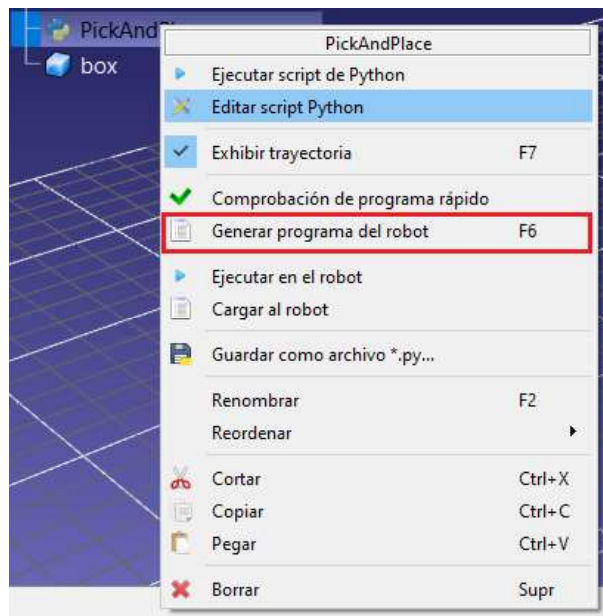
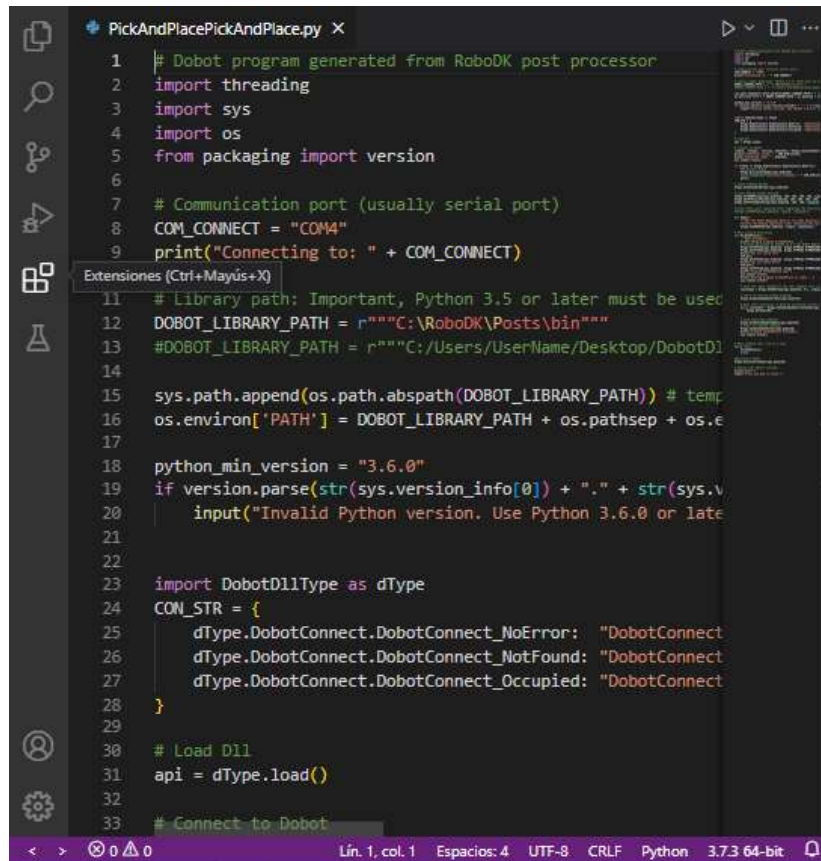


Figura 2.12 Cuadro de diálogo para generar el programa del robot.

Se exporta un archivo con el código propio que usa el robot, este código será colocado en el robot para validar que realice la tarea programada. En el capítulo 3 veremos imágenes del resultado de la simulación y verificación que el robot realiza la tarea asignada.



```
1 # Dobot program generated from RoboDK post processor
2 import threading
3 import sys
4 import os
5 from packaging import version
6
7 # Communication port (usually serial port)
8 COM_CONNECT = "COM4"
9 print("Connecting to: " + COM_CONNECT)
10
11 # Library path: Important, Python 3.5 or later must be used
12 DOBOT_LIBRARY_PATH = r"C:\RoboDK\Posts\bin"
13 #DOBOT_LIBRARY_PATH = r"C:/Users/UserName/Desktop/DobotD
14
15 sys.path.append(os.path.abspath(DOBOT_LIBRARY_PATH)) # temp
16 os.environ['PATH'] = DOBOT_LIBRARY_PATH + os.pathsep + os.e
17
18 python_min_version = "3.6.0"
19 if version.parse(str(sys.version_info[0]) + "." + str(sys.v
20     input("Invalid Python version. Use Python 3.6.0 or late
21
22
23 import DobotDllType as dType
24 CON_STR = {
25     dType.DobotConnect.DobotConnect_NoError: "DobotConnect
26     dType.DobotConnect.DobotConnect_NotFound: "DobotConnect
27     dType.DobotConnect.DobotConnect_Occupied: "DobotConnect
28 }
29
30 # Load Dll
31 api = dType.load()
32
33 # Connect to Dobot
```

Figura 2.13 Código propio del robot exportado de roboDK

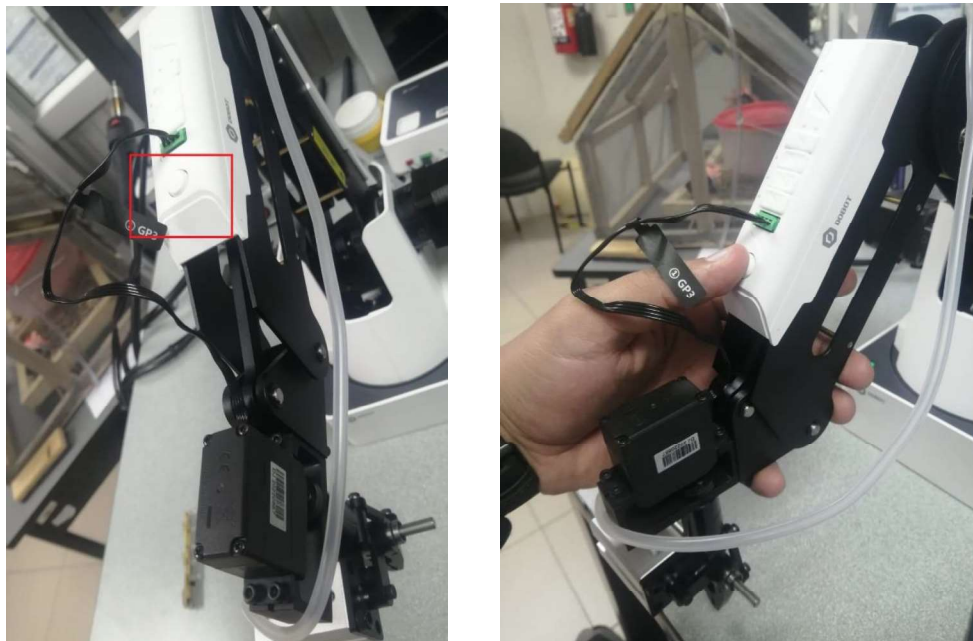
# CAPÍTULO 3

## 3. RESULTADOS Y ANÁLISIS

Para los resultados se comparó las dos principales técnicas de Machine Learning para programación de robots industriales como son:

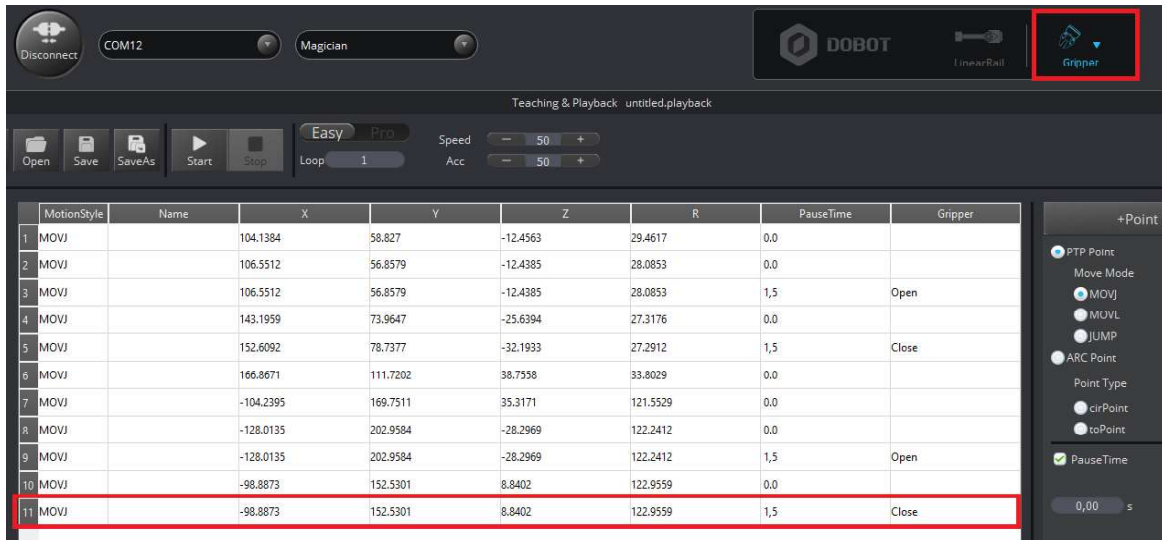
### 3.1 Aprendizaje supervisado.

La cual se programó al robot usando la técnica de **Teach Pendant**



**Figura 3.1 Botón de modo Teach en el robot Dobot Magician**

Como se observa en la figura 3.1 la programación por aprendizaje supervisado se consigue entrenando al robot dándole información conforme se mueve de posición usando el modo Teach, todos estos movimientos se irán guardando automáticamente gracias al software del robot dando como resultado el siguiente conjunto de posiciones.



**Figura 3.2 Programación usando el modo Teach usando el robot Dobot Magician.**

Tal como se observa este modo es totalmente manual, toda la información de los movimientos paso a paso se guarda en la interfaz del software del robot Dobot para luego decirle que repita todos estos pasos y completar la tarea del pick and place.

Algo que se puede destacar para este caso es que se está trabajando con un robot pequeño, sin embargo, si se deseara trabajar con un robot con mayores dimensiones este método sería un poco problemático.

### 3.2 Aprendizaje por reforzamiento Q-Learning

Para este método fue necesario adquirir los límites de las articulaciones del robot, y luego a obtener todos los pasos que puede realizar el robot tal como se describen en las siguientes tablas.

Articulación 1	Articulación 2	Articulación 3	Articulación 4
$\Theta 1$	$\Theta 2$	$\Theta 3$	$\Theta 4$
-90	-0.001	-10	-90

**Tabla 3.1 Límites negativos de las articulaciones del Dobot Magician**

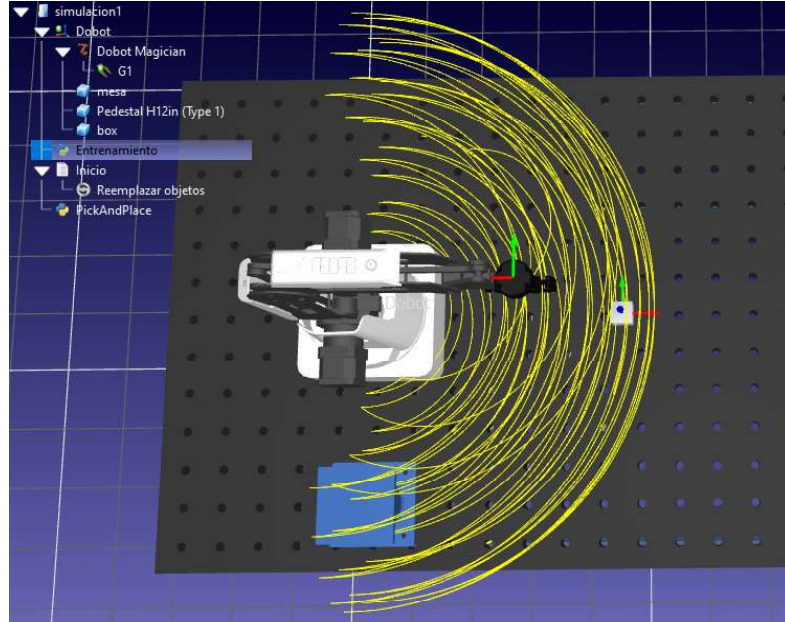
Articulación 1	Articulación 2	Articulación 3	Articulación 4
$\Theta 1$	$\Theta 2$	$\Theta 3$	$\Theta 4$
90	85	95	90

**Tabla 3.2 3.3 Límites positivos de las articulaciones del Dobot Magician**

A1	A2	A3	A4
-90	-0.001	-10	-90
-4.5	21.24925	16.25	-45
0	42.4995	42.50	0
45	63.74975	68.75	45
90	85	95	90

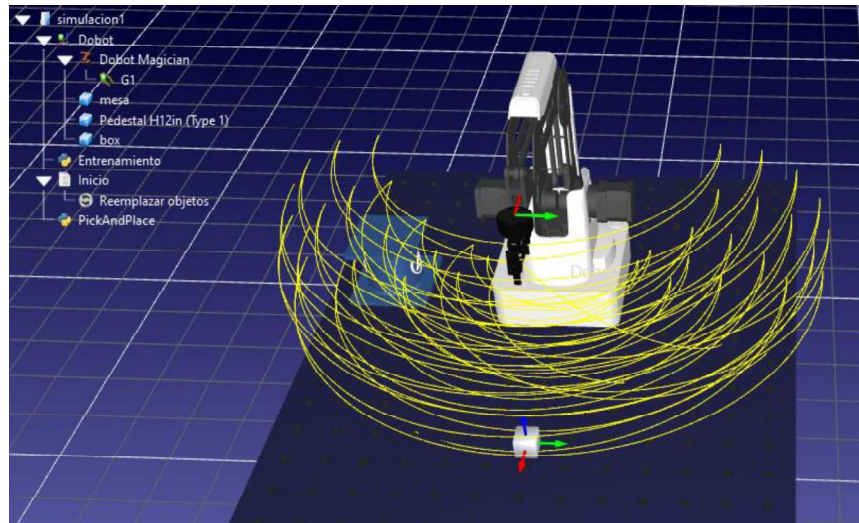
**Tabla 3.4 Posiciones de cada articulación**

La tabla 3.4 muestra los resultados que se obtuvieron simulando los pasos que puede realizar el robot en su entorno, esto con el fin de familiarizarse para luego poder decidir qué camino es el más efectivo, evitando los obstáculos y realizando la tarea que se le asigna, los resultados se muestran en la siguiente imagen.



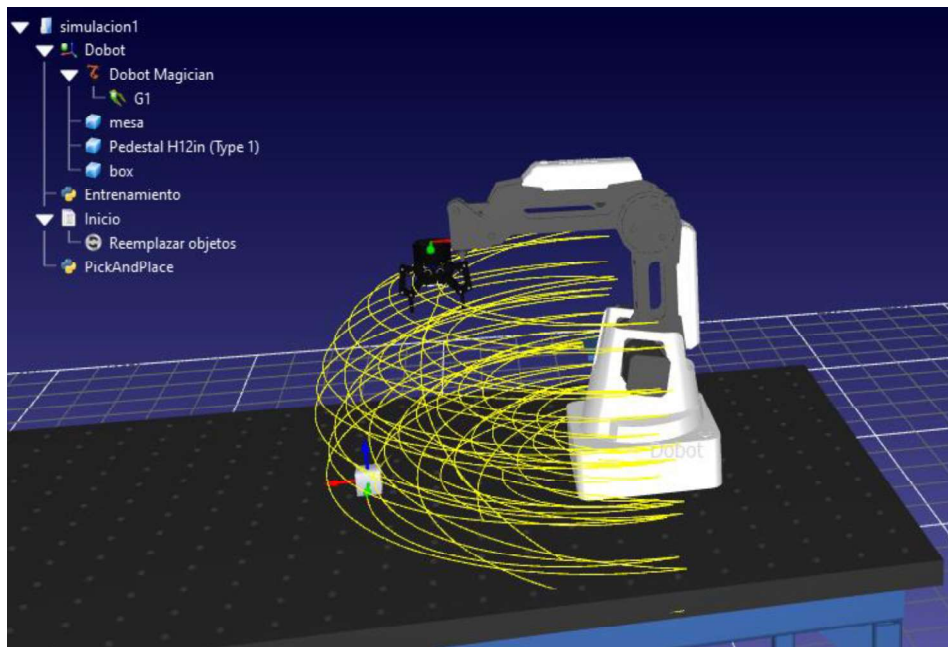
**Figura 3.3 Vista superior del espacio de trabajo del robot.**

Lo que podemos destacar es que las líneas amarillas son los caminos que puede alcanzar el robot sin ningún problema, mostrando que puede alcanzar su objetivo que es la caja y llevarla hasta su destino que será sobre el pedestal.



**Figura 3.4 Vista frontal del espacio de trabajo del robot.**





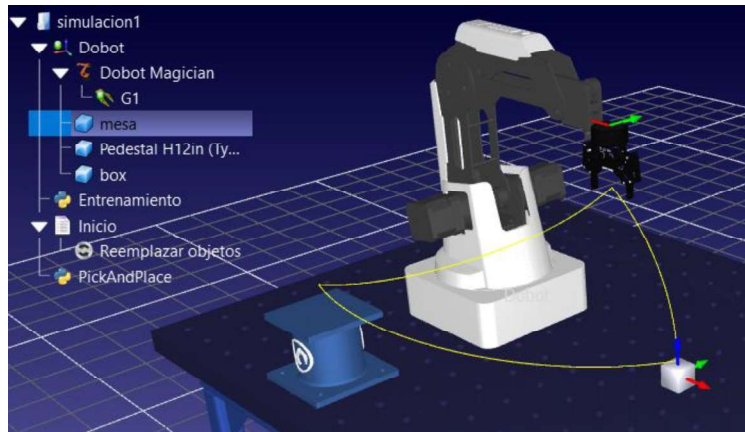
**Figura 3.5 Vista lateral del espacio de trabajo del robot.**

Con el espacio de trabajo ya identificado, se procedió a realizar la tarea designada al robot, que es un pick and place, esto se lo realiza utilizando los siguientes puntos que serán las coordenadas de inicio y final que tomará el robot.

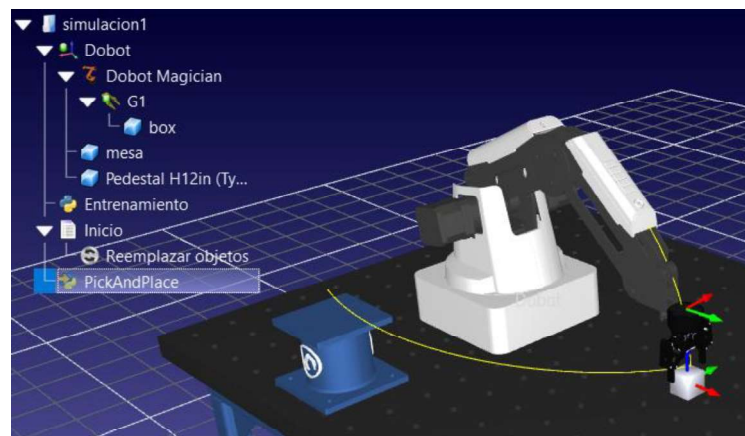
<i>Inicio</i>	<i>Fin</i>
0.11	-89.98
64.10	19.81
39.35	36.28
84.11	-5.98

**Tabla 3.5 Coordenadas de inicio y fin para el pick and place**

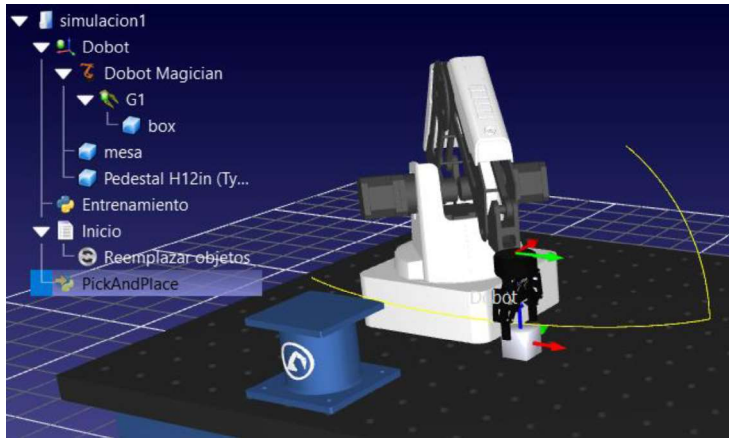
Como resultado de la simulación tenemos las siguientes imágenes donde nos muestran los pasos que sigue el robot, desde tomar la caja hasta dejarla en el destino, sobre el pedestal.



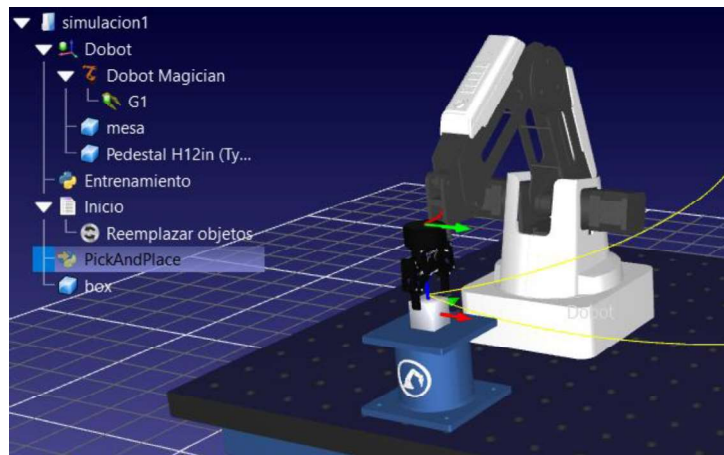
**Figura 3.6 Posición inicial del robot**



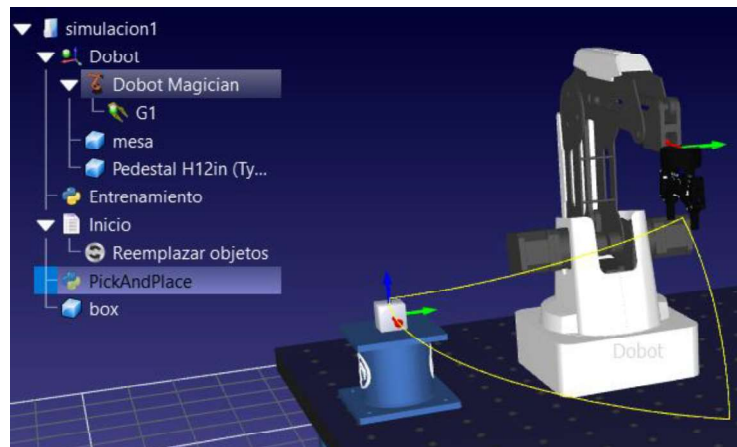
**Figura 3.7 Resultado de simulación robot sujetando la caja**



**Figura 3.8 Resultado de la simulación, Robot trasladando la caja hasta su destino**



**Figura 3.9 Resultado de la simulación Robot dejando la caja sobre el pedestal.**

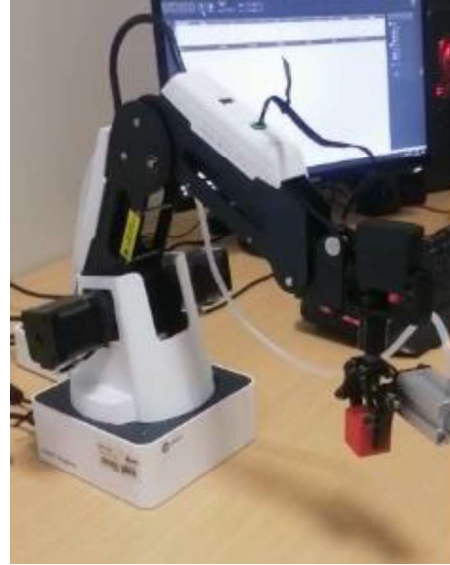


**Figura 3.10 Robot regresando a su posición de origen**

Con la simulación realizada con éxito y el robot realizo la tarea asignada se procedió a generar el programa propio del robot, y cargarlo para observar los movimientos y validar si se debe realizar algún cambio o corrección en la programación.



**Figura 3.11 Robot tomando la caja.**



**Figura 3.12 Robot moviéndose hasta el destino**



**Figura 3.13 Robot dejando el objeto en el destino.**

### 3.3 Comparación entre los métodos utilizados

Como podemos observar en ambos métodos el robot realiza la misma tarea que es de tomar un objeto y dejarla en otro lugar (“pick and place”), para conocer cual resulta más efectivo tomemos como comparación la siguiente tabla.

Método	Tiempo [Horas]	Facilidad	Equipos	Dinero \$
Teach	1:40	Se necesita mover el robot	Se necesita el Software del robot y El robot físico	2500\$
Aprendizaje por reforzamiento	00:40	Puede hacerse en simulación	El software de RoboDK	250\$

**Tabla 3.6 Comparación entre los métodos de programación estudiados.**

Al trabajar en la industria, los robots deben ser de grandes dimensiones para cumplir con todas las tareas asignadas, de la misma forma esto demanda más tiempo y esfuerzo si se desea programar o configurar este tipo de robots usando el método Teach, además al ser un método manual corre el riesgo de tener fallas o errores en la configuración.

Muy diferente es usar el método de aprendizaje por reforzamiento, que a diferencia del otro método se lo puede realizar en un entorno simulado y con más precisión ya que el robot primero se familiariza con su entorno y luego verifica qué acciones puede realizar o hasta qué puntos puede llegar.

Referente a los costos se puede notar que el aprendizaje por reforzamiento es más económico ya que solo se necesita el software de simulación es decir adquirir una licencia, a diferencia del otro método que es muy necesario tener el software del robot y el robot físico para realizar la programación.

### 3.4 Comparación con una actividad real en la industria.



Figura 3.14 Proceso de empaquetado de Huevos usando un robot industrial.

<b>Programación</b>	<b>Tiempo de mantenimiento[horas]</b>	<b>\$ dólares</b>
<b>Tradicional</b>	<b>1:30:00</b>	<b>\$1890</b>
<b>Utilizando aprendizaje por reforzamiento</b>	<b>0:40:00</b>	<b>\$840</b>
<b>Diferencia</b>	<b>0:50:00</b>	<b>\$1050</b>

Tabla 3.7 Comparación entre el método Teach y el aprendizaje por reforzamiento

Como se puede observar en el mundo laborar las industrias cada vez adquieren más equipos robóticos como es el caso de la industria empaquetadora de huevos tal como se muestra en la figura 3.11, lo que aquí llama la atención es que si se desea realizar un mantenimiento o una corrección a la rutina del robot, la empresa se ve obligado a parar la producción y ejecutar el empaquetamiento de forma manual, perdiendo así tiempo y corriendo el riesgo que el personal rompa el producto ya que es frágil.

Este análisis se ve comparado en la tabla 3.7, donde se puede observar el resultado de ejecutar un mantenimiento usando el método tradicional (método Teach), a diferencia de usar el método de aprendizaje por reforzamiento.

# CAPÍTULO 4

## 4. CONCLUSIONES Y RECOMENDACIONES

### Conclusiones

Dentro del Machine learning tenemos diferentes técnicas de aprendizaje de las cuales realizamos una comparación entre el aprendizaje supervisado y el aprendizaje por reforzamiento, mostrando los resultados de la tabla 3.6, donde podemos destacar que para el aprendizaje por reforzamiento tenemos las siguientes ventajas; no es necesario tener el robot físico para programar, la programación puede realizarse simulada y el tiempo que se toma configurar el robot es más corto que la programación tradicional.

Se utilizó el robot Dobot magician para el entrenamiento y validación del sistema, este robot tiene un área de trabajo limitada ya que al tener solo 4 articulaciones puede moverse hacia arriba, hacia abajo, hacia adelante o hacia atrás, hacia la derecha o hacia la izquierda, y girar en el plano horizontal. Referente a su entorno el robot moverá una caja de 2.5 cm por lado y la moverá por su entorno hasta llevarla a un lugar que se define en la programación en este caso sobre el pedestal, estos elementos se los puede observar en la figura 2.2.

Gracias al Q-learning se logró obtener el código con el cual el robot puede moverse por su área de trabajo luego de elegir la mejor opción basado en el entrenamiento, cumpliendo con la tarea asignada que fue realizar el “Pick and Place”, los resultados de la simulación lo podemos verificar desde la figura 3.6 hasta la 3.10 donde se muestra los movimientos que realiza el robot. Se debe destacar que este sistema puede ser usado en diferentes robots ya que el entrenamiento se centra en el lugar de trabajo del robot es decir los puntos en el que el robot puede moverse.



## **Recomendaciones**

Cuando se genere el código propio del robot, es necesario revisar por si se agregan funciones que no son utilizadas por el robot, pero sí por el software RoboDK y desde luego el lenguaje Python.

Tener muy en cuenta la cantidad de articulaciones de cada robot y de sus límites ya que el sistema de entrenamiento utiliza estos datos para ayudar a entrenar al robot, de esta forma se previenen errores.

Investigar y estudiar las funciones y el lenguaje Python que usa roboDK con el fin de realizar tareas más complejas y no tener inconvenientes al momento de simular el código o la tarea que le estemos asignando al robot.

# BIBLIOGRAFÍA

- Dobot. (2018). *Dobot Magician User Manual*. Shenzhen Yuejiang Technology Co., Ltd.
- Dobot. (s.f.). *Dobot*. Obtenido de [https://www.dobot.com.mx/dobot\\_magician](https://www.dobot.com.mx/dobot_magician)
- Heras, J. M. (10 de octubre de 2020). *IArtificial.net*. Obtenido de <https://www.iartificial.net/librerias-de-python-para-machine-learning/>
- IBM. (s.f.). *IBM*. Obtenido de <https://www.ibm.com/mx-es/analytics/machine-learning>
- industrial, C. d. (2018). *aula21*. Obtenido de <https://www.cursosaula21.com/como-funciona-la-robotica-industrial/>
- INTEL. (Marzo de 2020). *INTEL*. Obtenido de <https://www.intel.la/content/www/xl/es/robotics/robotic-arm.html#:~:text=Adem%C3%A1s%20de%20las%20juntas%20rotativas,y%20los%20componentes%20de%20software.>
- Moya, R. (18 de ABRIL de 2022). *GITHUB*. Obtenido de [https://github.com/RicardoMoya/Reinforcement\\_Learning\\_with\\_Python/blob/master/1\\_Aprendizaje\\_Por\\_Refuerzo.ipynb](https://github.com/RicardoMoya/Reinforcement_Learning_with_Python/blob/master/1_Aprendizaje_Por_Refuerzo.ipynb)
- python, R. (s.f.). *RoboDK*. Obtenido de <https://robodk.com/doc/en/PythonAPI/robodk.html#robodk.robotlink.Robolink.ItemList>
- RoboDK. (s.f.). *RoboDK*. Obtenido de <https://robodk.com/doc/es/Robot-Programs.html#:~:text=RoboDK%20es%20un%20simulador%20enfocado,y%20un%20controlador%20de%20robot.>
- Robots, R. d. (2 de noviembre de 2022). *Revistas de Robots*. Obtenido de <https://revistaderobots.com/robots-y-robotica/robots-industriales-y-robotica-industrial/>
- Rus, D. (2015). Robótica: una década de transformaciones. En D. Rus, *Robótica: una década de transformaciones*.
- School, E. B. (5 de Abril de 2019). *Esneca Business School*. Obtenido de <https://www.esneca.com/blog/brazo-robotico-industrias/>

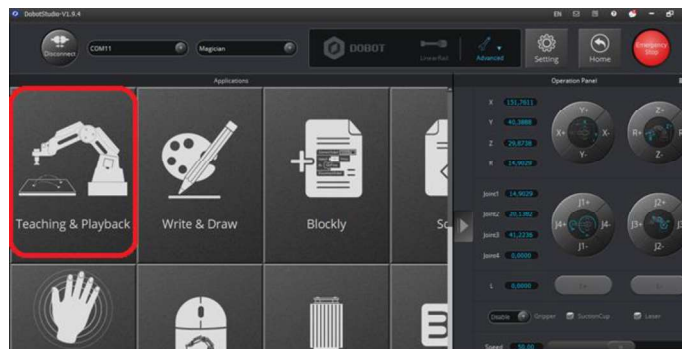
TEAM, D. S. (2020). *DATA SCIENCE*. Obtenido de <https://datascience.eu/es/aprendizaje-automatico/q-learning/>

# APÉNDICES

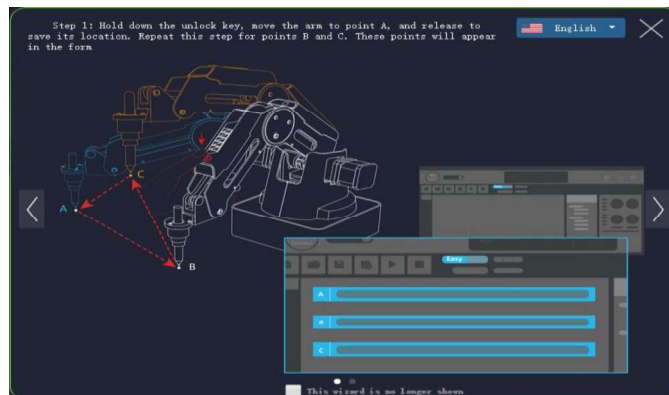
## PRACTICA #1 TEACHING & PLAYBACK USANDO EL ROBOT: DOBOT MAGICIAN

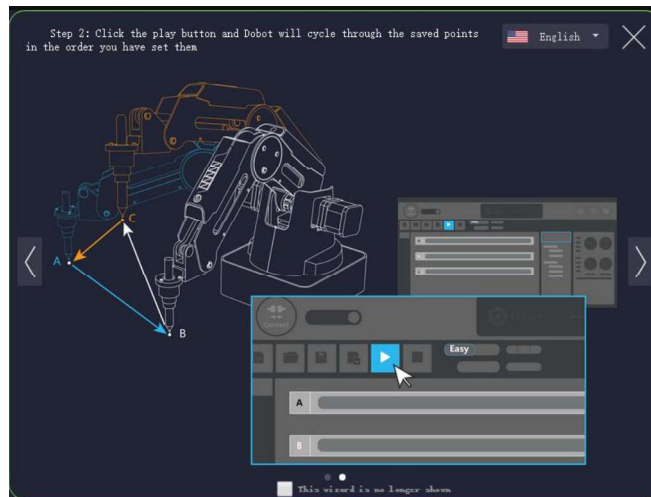
Se realizará una programación usando el método Teaching & Playback para que el robot haga una rutina de pick and place, es decir tome un objeto de un lugar y lo coloque en otra posición, para esto seguiremos los siguientes pasos.

1. Abrimos el software DobotStudio y seleccionamos la opción #1 Teaching & Playback para realizar la programación del robot.

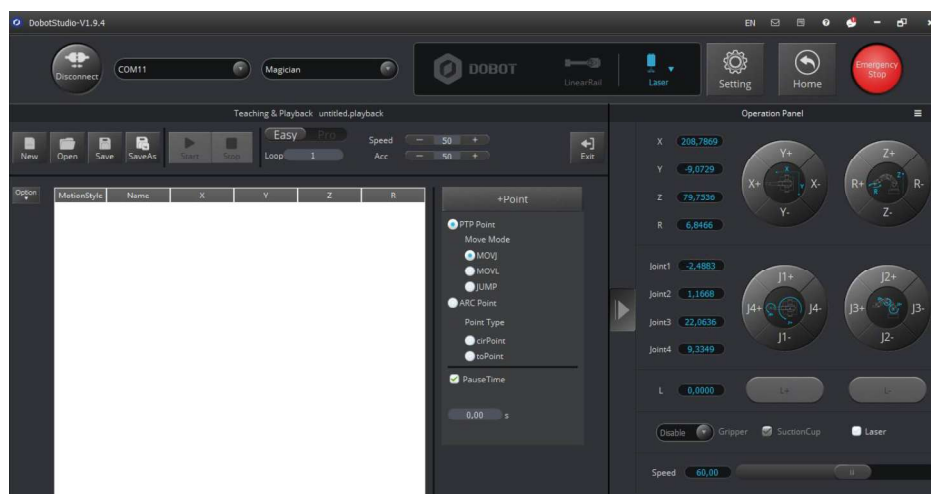


2. Nos aparecerá un pequeño tutorial que nos indica cómo se programa el robot para que copie los pasos que se le configura manualmente.

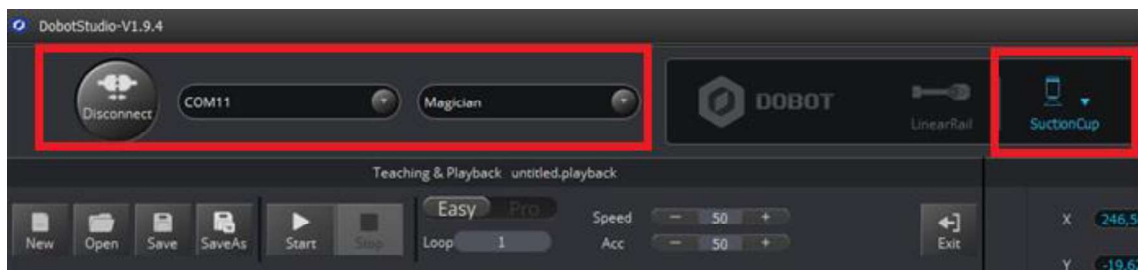




3. Luego de cerrar el tutorial aparecerá la pantalla para comenzar a indicarle al robot que pasos debe seguir.



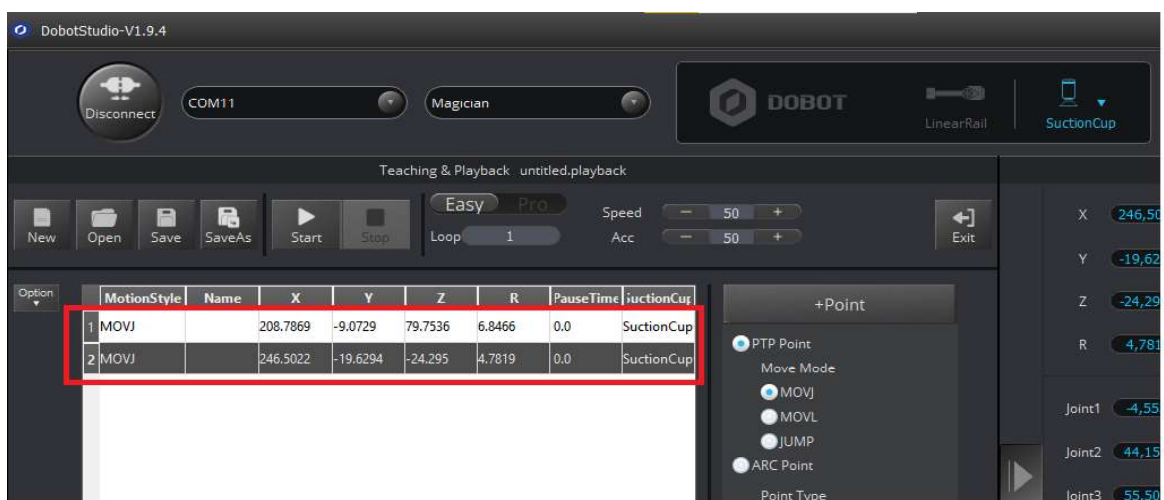
4. Verificamos que el robot se encuentre conectado al software y que en las opciones de operación se encuentre en SUCTIONCUP.



5. Para realizar los movimientos libres con el robot primero mantenemos pulsado el botón que aparece como un candado sobre el brazo del robot.



6. Al momento de soltar el botón, se guardará la posición y acción del robot en las líneas de instrucciones, además la posición también se verá en el panel de operación.



The screenshot shows the DobotStudio-V1.9.4 interface. At the top, there are connection settings for 'COM11' and 'Magician'. The main area contains a 'Teaching & Playback' section with a table of motion instructions. The table has columns for 'MotionStyle', 'Name', 'X', 'Y', 'Z', 'R', 'PauseTime', and 'JuctionCup'. Two rows are highlighted with a red box:

MotionStyle	Name	X	Y	Z	R	PauseTime	JuctionCup
1	MOVJ	208.7869	-9.0729	79.7536	6.8466	0.0	SuctionCup
2	MOVJ	246.5022	-19.6294	-24.295	4.7819	0.0	SuctionCup

Below the table, there is a '+Point' section with radio buttons for 'PTP Point', 'Move Mode' (with sub-options MOVJ, MOVL, JUMP), 'ARC Point', and 'Point Type'. On the right side, there is a panel showing real-time coordinates for X, Y, Z, R, Joint1, Joint2, and Joint3.



Siguiendo la forma de guardar los pasos, inicialmente vamos a guardar dos posiciones.

- a. Guardamos la posición inicial del movimiento, que será como se observa en la siguiente imagen.



En la línea de instrucciones se guardará de forma automática el movimiento registrado.

	MotionStyle	Name	X	Y	Z	R	PauseTime	SuctionCup
1	MOVJ		208.7869	-9.0729	79.7536	6.8466	0.0	SuctionCup

- b. De igual forma agregamos otra posición, moviendo el robot con nuestras manos a una posición cercada a la superficie de la mesa de trabajo.

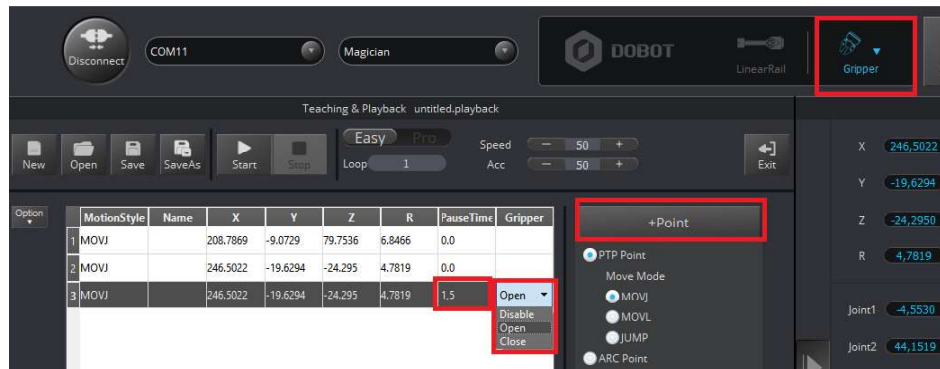


	MotionStyle	Name	X	Y	Z	R	PauseTime	SuctionCup
1	MOVJ		208.7869	-9.0729	79.7536	6.8466	0.0	SuctionCup
2	MOVJ		246.5022	-19.6294	-24.295	-4.7819	0.0	SuctionCup

7. Ahora para activar las pinzas, en este caso para que se abran, cambiamos el modo de operación del robot a **GRIPPER**, luego para no mover el brazo damos clic en +Point para agregar otra posición a la línea de instrucciones, pero ahora se agregara con la opción de gripper.

Luego configuramos la acción del gripper la colocamos en "Open" y también el tiempo que estará activo para que la pinza abra totalmente le colocamos 1.5 segundos.





- Ahora agregamos un punto más, con las pinzas abiertas acercándolas al objeto y con el tipo de acción SUCTIONCUP.

Teaching & Playback - untitled.playback

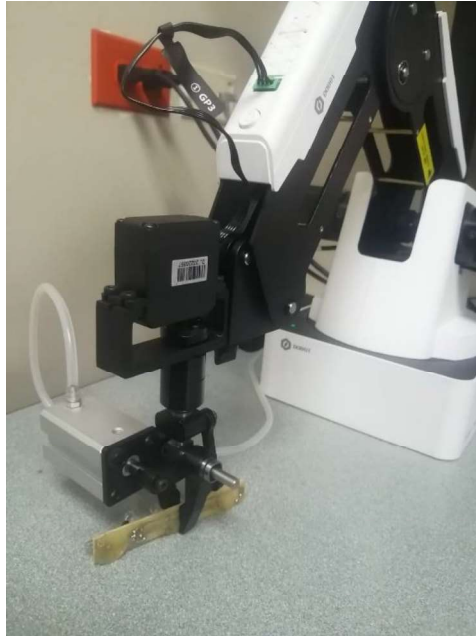
MotionStyle	Name	X	Y	Z	R	PauseTime	SuctionCup
1	MOVJ	104.1304	50.027	-12.4563	29.4617	0.0	SuctionCupOff
2	MOVJ	106.5512	56.8579	-12.4385	28.0853	0.0	SuctionCupOff
3	MOVJ	106.5512	56.8579	-12.4385	28.0853	1.5	SuctionCupOff
4	MOVJ	143.1959	73.9647	-25.6394	27.3176	0.0	SuctionCupOff



9. Agregamos otro punto usando +Point para que la posición actual no se cambie, pero lo haremos en modo GRIPPER nuevamente ya que será para cerrar las pinzas.

Teaching & Playback - untitled.playback

MotionStyle	Name	X	Y	Z	R	PauseTime	Gripper
1	MOVJ	104.1384	58.827	-12.4563	29.4617	0.0	
2	MOVJ	106.5512	56.8579	-12.4385	28.0853	0.0	
3	MOVJ	106.5512	56.8579	-12.4385	28.0853	1.5	Open
4	MOVJ	143.1959	73.9647	-25.6394	27.3176	0.0	
5	MOVJ	152.6092	78.7377	-32.1933	27.2912	1.5	Close



10. Ahora, Agregamos otro movimiento, pero en modo SuctionCup, ahora será para levantar el brazo, levantando el objeto.

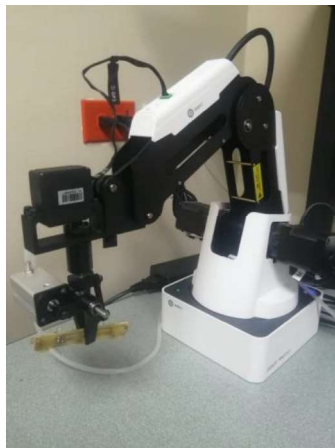
Teaching & Playback - untitled.playback

Easy Pro Speed 50 Acc 50

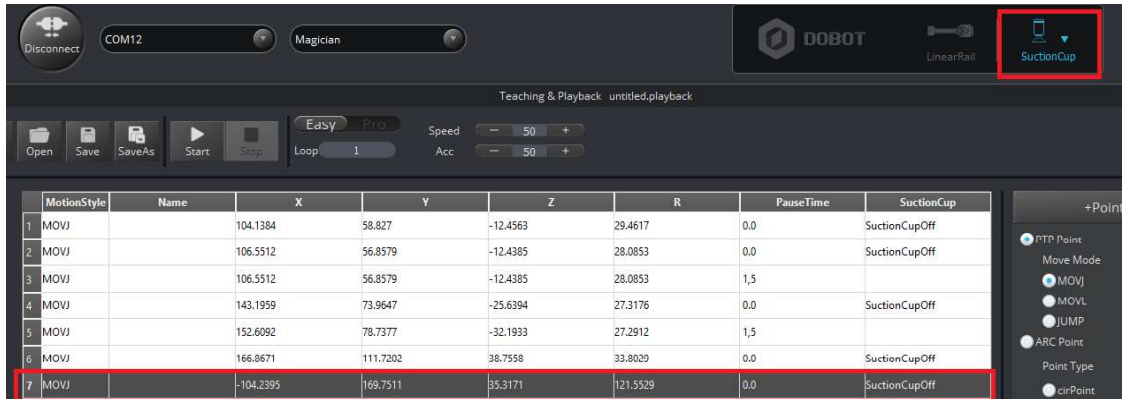
Option	MotionStyle	Name	X	Y	Z	R	PauseTime	SuctionCup
1	MOVJ		208.7869	-9.0729	79.7536	6.8466	0.0	SuctionCup
2	MOVJ		246.5022	-19.6294	-24.295	4.7819	0.0	SuctionCup
3	MOVJ		246.5022	-19.6294	-24.295	4.7819	1,5	
4	MOVJ		246.5022	-19.6294	-24.295	4.7819	1,5	
5	MOVJ		221.8598	-17.4608	53.4894	4.8348	0.0	SuctionCup

PTP Point  
 Move Mode  
 MOVJ  
 MOVL  
 JUMP  
 ARC Point

X: 221,8598  
 Y: -17,4608  
 Z: 53,4894  
 R: 4,8348  
 Joint1: -4,5000  
 Joint2: 12,0884

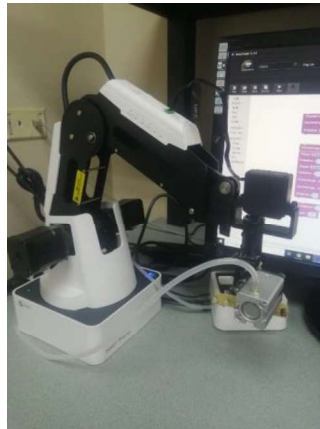


11. Siguiendo la misma idea moveremos el robot a una posición aleatoria que escojamos, la cual será cerca de donde queremos dejar el objeto, preferible sobre la posición final.

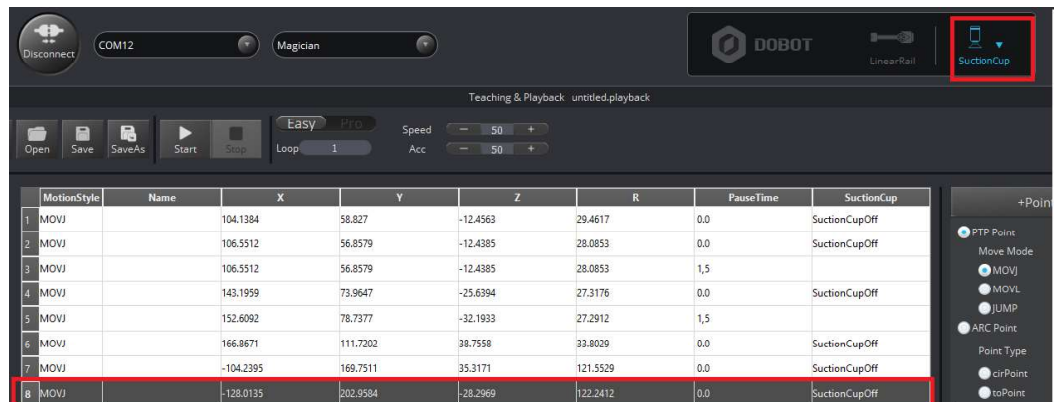


The screenshot shows the DOBOT software interface. At the top, there are controls for 'Disconnect', 'COM12', 'Magician', and 'DOBOT'. A 'SuctionCup' button is highlighted with a red box. Below this is a 'Teaching & Playback' section with 'untitled.playback' and controls for 'Easy', 'Pro', 'Speed', and 'Acc'. A table of motion points is displayed, with the 7th point highlighted in red. The table has columns for MotionStyle, Name, X, Y, Z, R, PauseTime, and SuctionCup.

MotionStyle	Name	X	Y	Z	R	PauseTime	SuctionCup
1	MOVJ	104.1384	58.827	-12.4563	29.4617	0.0	SuctionCupOff
2	MOVJ	106.5512	56.8579	-12.4385	28.0853	0.0	SuctionCupOff
3	MOVJ	106.5512	56.8579	-12.4385	28.0853	1,5	
4	MOVJ	143.1959	73.9647	-25.6394	27.3176	0.0	SuctionCupOff
5	MOVJ	152.6092	78.7377	-32.1933	27.2912	1,5	
6	MOVJ	166.8671	111.7202	38.7558	33.8029	0.0	SuctionCupOff
7	MOVJ	-104.2395	169.7511	35.3171	121.5529	0.0	SuctionCupOff



12. Luego acercamos el robot a la superficie de trabajo, y se guardara ese punto también en modo SuctionCup.



The screenshot shows the DOBOT software interface. At the top, there are controls for 'Disconnect', 'COM12', 'Magician', and 'DOBOT'. A 'SuctionCup' button is highlighted with a red box. Below this is a 'Teaching & Playback' section with 'untitled.playback' and controls for 'Easy', 'Pro', 'Speed', and 'Acc'. A table of motion points is displayed, with the 8th point highlighted in red. The table has columns for MotionStyle, Name, X, Y, Z, R, PauseTime, and SuctionCup.

MotionStyle	Name	X	Y	Z	R	PauseTime	SuctionCup
1	MOVJ	104.1384	58.827	-12.4563	29.4617	0.0	SuctionCupOff
2	MOVJ	106.5512	56.8579	-12.4385	28.0853	0.0	SuctionCupOff
3	MOVJ	106.5512	56.8579	-12.4385	28.0853	1,5	
4	MOVJ	143.1959	73.9647	-25.6394	27.3176	0.0	SuctionCupOff
5	MOVJ	152.6092	78.7377	-32.1933	27.2912	1,5	
6	MOVJ	166.8671	111.7202	38.7558	33.8029	0.0	SuctionCupOff
7	MOVJ	-104.2395	169.7511	35.3171	121.5529	0.0	SuctionCupOff
8	MOVJ	-128.0135	202.9584	-28.2969	122.2412	0.0	SuctionCupOff



13. Agregamos otro punto usando +Point para no cambiar la posición actual del robot, usaremos este punto para abrir nuevamente las pinzas y dejar el objeto en la superficie de trabajo, recordar que se deme agregar este punto en modo Gripper.

También en las opciones del Gripper colocamos en modo open y en el tiempo 1,5 segundos.

The screenshot shows the Dobot software interface. At the top right, the 'Gripper' button is highlighted with a red box. Below the table, the '+Point' button is also highlighted with a red box. The table contains the following data:

MotionStyle	Name	X	Y	Z	R	PauseTime	Gripper
1	MOVJ	104.1384	58.827	-12.4563	29.4617	0.0	
2	MOVJ	106.5512	56.8579	-12.4385	28.0853	0.0	
3	MOVJ	106.5512	56.8579	-12.4385	28.0853	1,5	Open
4	MOVJ	143.1959	73.9647	-25.6394	27.3176	0.0	
5	MOVJ	152.6092	78.7377	-32.1933	27.2912	1,5	Close
6	MOVJ	166.8971	111.7202	98.7338	33.8029	0.0	
7	MOVJ	-104.2395	169.7511	35.3171	121.5529	0.0	
8	MOVJ	-128.0135	202.9584	-28.2969	122.2412	0.0	
9	MOVJ	-128.0135	202.9584	-28.2969	122.2412	1,5	Open



Teaching & Playback untitle.playback

COM12    Magician    DOBOT    Linear/Rail    SuctionCup

Open Save SaveAs Start Stop    Easy Pro    Speed 50    Acc 50

MotionStyle	Name	X	Y	Z	R	PauseTime	SuctionCup
1	MOVJ	104.1304	58.827	-12.4563	29.4617	0.0	SuctionCupOff
2	MOVJ	106.5512	56.8579	-12.4385	28.0853	0.0	SuctionCupOff
3	MOVJ	106.5512	56.8579	-12.4385	28.0853	1,5	
4	MOVJ	143.1959	73.9647	-25.6394	27.3176	0.0	SuctionCupOff
5	MOVJ	152.6092	78.7377	-32.1933	27.2912	1,5	
6	MOVJ	166.8671	111.7202	38.7558	33.8029	0.0	SuctionCupOff
7	MOVJ	-104.2395	169.7511	35.3171	121.5529	0.0	SuctionCupOff
8	MOVJ	128.0135	202.9584	28.2969	122.2412	0.0	SuctionCupOff
9	MOVJ	-128.0135	202.9584	-28.2969	122.2412	1,5	
10	MOVJ	-98.8873	152.5301	8.8402	122.9559	0.0	SuctionCupOff

+Point

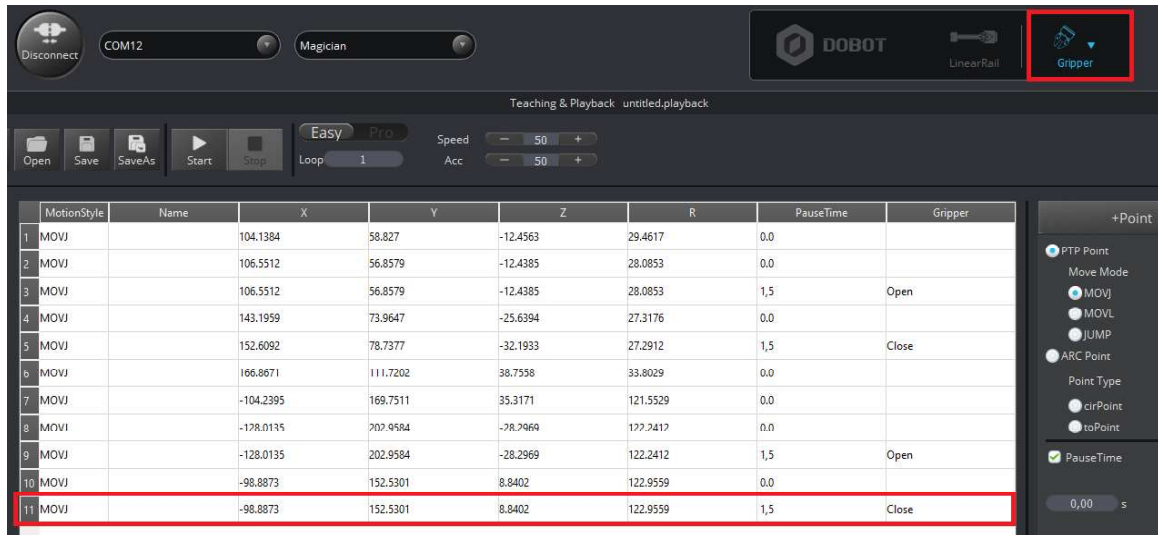
- PTP Point
- Move Mode
  - MOVJ
  - MOVL
  - JJUMP
  - ARC Point
- Point Type
  - cirPoint
  - toPoint
- PauseTime

0,00 s

14. Agregamos otro punto para levantar el robot sobre el objeto para poder cerrar las pinzas y que estas no agarren el objeto nuevamente.



15. Agregamos otro punto, pero esta vez usando nuevamente +Point para no perder la posición actual del robot, este lo agregamos para poder cerrar las pinzas.



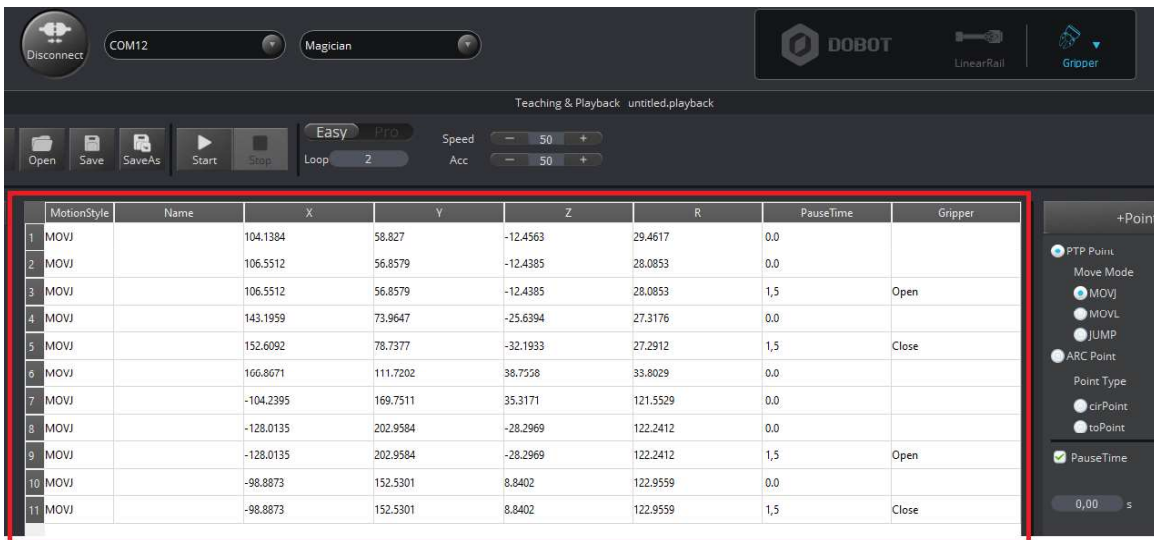
The screenshot shows the DOBOT software interface. At the top, there are controls for 'Disconnect', 'COM12', and 'Magician'. The 'Gripper' button is highlighted with a red box. Below this is a 'Teaching & Playback' section with 'untitled.playback' and controls for 'Easy/Pro' modes, 'Speed', and 'Acc'. A table of motion points is displayed, with the 11th point highlighted in red. To the right, the '+Point' panel shows 'PTP Point' selected, 'Move Mode' set to 'MOVJ', and 'PauseTime' set to 0.00 s.

	MotionStyle	Name	X	Y	Z	R	PauseTime	Gripper
1	MOVJ		104.1384	58.827	-12.4563	29.4617	0.0	
2	MOVJ		106.5512	56.8579	-12.4385	28.0853	0.0	
3	MOVJ		106.5512	56.8579	-12.4385	28.0853	1,5	Open
4	MOVJ		143.1959	73.9647	-25.6394	27.3176	0.0	
5	MOVJ		152.6092	78.7377	-32.1933	27.2912	1,5	Close
6	MOVJ		166.8671	111.7202	38.7558	33.8029	0.0	
7	MOVJ		-104.2395	169.7511	35.3171	121.5529	0.0	
8	MOVJ		-178.0135	202.9584	-28.2969	122.2412	0.0	
9	MOVJ		-128.0135	202.9584	-28.2969	122.2412	1,5	Open
10	MOVJ		-98.8873	152.5301	8.8402	122.9559	0.0	
11	MOVJ		-98.8873	152.5301	8.8402	122.9559	1,5	Close

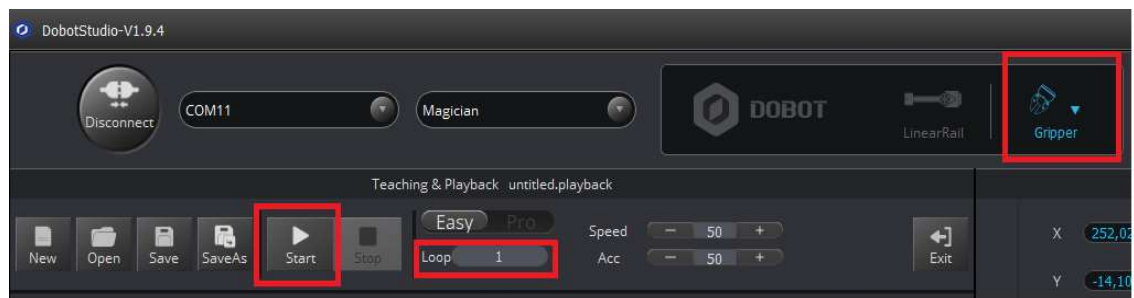
Hay que recordar que para usar las pinzas debemos estar en modo GRIPPER y además en las opciones del gripper colocar la opción "close", en el tiempo colocamos 1,5 segundos tal como muestra la imagen.



16. Para la simulación tendremos la rutina del robot parecida a la siguiente.



17. Procedemos a configurar la cantidad de veces que queremos que se repita la rutina del robot, esto lo hacemos en la casilla de "LOOP", adicional verificamos que el modo de operación se encuentre en "GRIPPER" si todo lo anterior

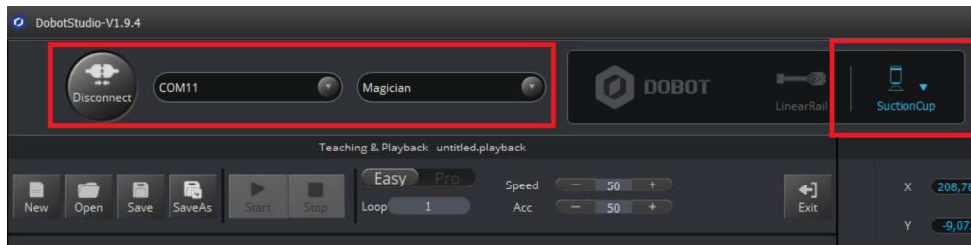


mencionado está listo daremos clic en "Start" y el robot comenzara a realizar los pasos programados.



## ERRORES COMUNES.

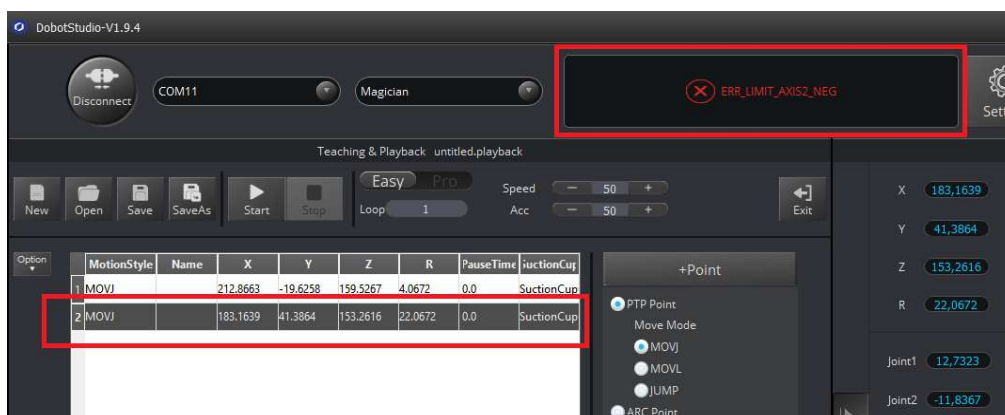
- Verificar que el robot se encuentre conectado de lo contrario no se podrá registrar la posición en la que se encuentra y por tanto tampoco se podrán ir agregando los puntos de forma manual a la rutina del robot.



- Si las pinzas no se activan verificar siempre que el modo de operación se encuentre en modo GRIPPER de lo contrario nunca se activaran.



- Por último, verificar que los puntos que escojamos de forma manual con el robot sean puntos permitidos, ya que si elevamos o abrimos mucho el brazo este nos mostrara un error indicando que ese punto no es valido.



Para solucionar este error solo borramos el punto que muestra error y buscamos uno nuevo, donde permita el robot.