REVISTA ELECTRO, Vol. 44 pp. 188-193, Oct 2022, Chihuahua, Chih. México
http:// electro.itchihuahua.edu.mx/revista
ISSN 1405-2172

SIMULADOR DE UN ROBOT DE 6GL USANDO LabVIEW Y PYTHON

Ignacio Javier Vázquez Cuevas [1], José Alejandro López Corella [2], Sergio Iván Hernández Ruiz [3], Martin Ochoa Alegria[4], Beatrice Pérez Arce [5]

Tecnológico Nacional de México / I.T. de Nogales, Metal Mecánica Ingeniería Mecatrónica, Eléctrica y Electrónica Ingeniería en Electrónica, Av. Tecnológico 911, Nogales, Sonora, México
+52(631) 311 1881. Ext 1129.
jose.lc@nogales.tecnm.mx

RESUMEN.

En el presente trabajo se demuestra el desarrollo de un simulador de Robot de 6 GDL utilizando LabVIEW, para la solución de los modelos matemáticos se utilizó Python con la librería de numpy para hacer operaciones con matrices multidimensionales. Las herramientas matemáticas para la solución de la cinemática directa se utilizó el Algoritmo de Denavit–Hartenberg , en el modelado de la cinemática inversa se utilizó Desacoplo Cinemático para encontrar la solución de posición y orientación, posteriormente se desarrolló el simulador del robot en LabVIEW con objetos 3D exportados de SolidWorks, de esta manera se pudo comparar el funcionamiento correcto del simulador en LabVIEW en la cual es una interfaz mucho más versátil y amigable para la creación de un prototipo físico del robot como futuro desarrollo ya que la simulación nos da la certeza del funcionamiento de la cinemática como proceso previo para la aplicación física del Robot de 6GDL.

Palabras Clave: Robot 6 GDL, LabVIEW, Python, SolidWorks.

ABSTRACT.

In the present work, the development of a 6 GDL Robot simulator using LabVIEW is demonstrated, for the solution of the mathematical models, Python was used with the numpy library to perform operations with multidimensional arrays. The mathematical tools for the solution of the direct kinematics were used the Denavit-Hartenberg Algorithm, in the modeling of the inverse kinematics Kinematic Decoupling was used to find the position and orientation solution, later the robot simulator was developed in LabVIEW with objects 3D exported from SolidWorks, in this way it was possible to compare the correct operation of the simulator in LabVIEW in which it is a much more versatile and friendly interface for the creation of a physical prototype of the robot as a future development since the simulation gives us the certainty of the operation of the kinematics as a previous process for the physical application of the 6GDL Robot.

Keywords: Robot 6 GDL, LabView, LabVIEW, Python, SolidWorks.

1. INTRODUCCIÓN

Existen muchas plataformas para el desarrollo de simuladores de robots articulados entre ellas se utilizan WOLFRAM MATHEMATICA para el modelado de las cinemática del robot de 6GDL [1], también se han desarrollado robots con el TOOLBOX ROBOTICS del software MATLAB [2] en la cual facilita la aplicación de los modelos matemáticos creando una interfaz interactiva, también se han desarrollado simuladores con otros lenguajes de programación como C# [3] utilizando librerías de OpenGL . Se han creado simuladores utilizando la plataforma LabVIEW con el módulo MathScript de NI, este módulo ejecuta los Script de Matlab [4].
El uso de Python no es la excepción para el desarrollo de robots, se ha utilizado este lenguaje para el desarrollo de robots móviles y robots articulados [5] [6] [7] [8].

En el presente trabajo se muestra el procedimiento del desarrollo del robot de 6 GDL integrando diferentes tecnologías facilitando el diseño del simulador en 3D, como se observa en la figura 1 primero se desarrolla los modelados matemáticos de cinemática directa e inversa con en Python, posteriormente se diseñó el robot en SolidWorks , se tiene que crear sub ensamblajes de cada parte móvil del robot, esto con el fin de crear el simulador usando estas partes móviles, y para finalizar se crea el simulador con LabVIEW , en este software se integran las modelos de las partes móviles creadas en SolidWorks y estas se manipulan con los modelos matemáticos creados en Python.

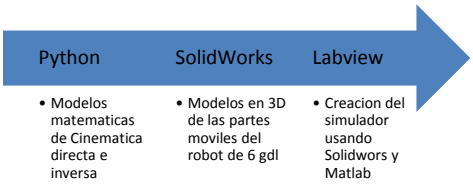


Figura 1.- Integración de software

2. CÁLCULOS MATEMÁTICOS.

Los cálculos matemáticos, como las matrices de transformación homogéneas y las ecuaciones cinemáticas, fueron programadas

con el lenguaje de programación de Python y la librería de *numpy* para hacer operaciones con matrices multidimensionales.

2.1 MATRICES DE ROTACIÓN.

Las matrices homogéneas de rotación son empleadas para rotar una figura en un sistema de origen XYZ en torno a uno de sus ejes, de modo que hay tres matrices distintas que representan la rotación para los ejes x, y y z en la fig. 2.

```python
import numpy as np

def matrizHomogeneaRotacionX(angulo):
    anguloRadianes = degToRad(angulo)
    rotacionX =  np.array([[1, 0,                    0,                    0],
                           [0, np.cos(anguloRadianes), -np.sin(anguloRadianes), 0],
                           [0, np.sin(anguloRadianes), np.cos(anguloRadianes),  0],
                           [0, 0,                    0,                    1]])
    return rotacionX

def matrizHomogeneaRotacionY(angulo):
    anguloRadianes = degToRad(angulo)
    rotacionY =  np.array([[np.cos(anguloRadianes), 0, np.sin(anguloRadianes), 0],
                           [0,                    1, 0,                    0],
                           [-np.sin(anguloRadianes),0, np.cos(anguloRadianes), 0],
                           [0,                    0, 0,                    1]])
    return rotacionY

def matrizHomogeneaRotacionZ(angulo):
    anguloRadianes = degToRad(angulo)
    rotacionZ =  np.array([[np.cos(anguloRadianes), -np.sin(anguloRadianes), 0, 0],
                           [np.sin(anguloRadianes),  np.cos(anguloRadianes), 0, 0],
                           [0,                       0,                     1, 0],
                           [0,                       0,                     0, 1]])
    return rotacionZ

def degToRad(angulo):
    anguloRad = angulo * (np.pi/180)
    return anguloRad
```

Fig. 2 Matrices de rotación en Python.

2.2 MATRIZ DE TRASLACIÓN.

La matriz de transformación homogénea de traslación pertenece al cálculo del traslado del sistema actual XY Z hasta el sistema destino XYZ, definido por la matriz en la fig. 3.

```python
import numpy as np

def matrizHomogeneaTraslacion(px, py, pz):

    traslacion = np.array([[1, 0, 0, px],
                           [0, 1, 0, py],
                           [0, 0, 1, pz],
                           [0, 0, 0, 1]])
    return traslacion
```

Fig. 3 Matriz de traslación en Python.

2.3 MATRIZ DE DENAVIT-HARTENBERG.

Con el método de representación de Denavit-Hartenberg se calcula la posición final del robot y también la de cada una de sus articulaciones, tal y como se observa en la fig. 4.

```python
import numpy as np
from matrizHomogeneaRotacion import *
from matrizHomogeneaTraslacion import matrizHomogeneaTraslacion

def matrizHomogeneaDH(L, A, D, T):
    matrizHomogeneaDH = matrizHomogeneaRotacionZ(T).dot(matrizHomogeneaTraslacion(0,0,D))
    matrizHomogeneaDH  = matrizHomogeneaDH.dot(matrizHomogeneaTraslacion(L,0,0))
    matrizHomogeneaDH  = matrizHomogeneaDH.dot(matrizHomogeneaRotacionX(A))
    return matrizHomogeneaDH
```

Fig. 4 Matriz de Denavit-Hartenberg en Python.

2.4 CINEMÁTICA DIRECTA

Se creó la función Directa6GDL para realiza los parámetros de las articulaciones del Robot 6 GDL. [9] [10] [11]

La función de cinemática directa para el robot de seis grados de libertad utiliza el mismo método que los otros elementos robóticos creados a lo largo del semestre: la convención de Denavit-Hartenberg. Se determinan las matrices de transformación homogénea de cada eslabón con los cuatro parámetros que requiere el método (ver fig. 5).
Al obtener todas las matrices de transformación homogéneas, una nueva matriz es creada con la posición de los eslabones a fin de ser empleados en la representación gráfica del robot en la GUI. Asimismo, se obtienen los cálculos de roll, pitch y yaw para ser mostrados también en la interfaz gráfica de usuario.

```python
import numpy as np
from matrizHomogeneaDH import *

def robot6GdlCD(L1, T1, L2, T2, L3, T3, L4, T4, L5, T5, L6, T6):
    H01 = matrizHomogeneaDH(0,90,L1,T1)
    H12 = matrizHomogeneaDH(L2,0,0,T2)
    H23 = matrizHomogeneaDH(0,90,0,T3)
    H34 = matrizHomogeneaDH(0,-90,L4,T4)
    H45 = matrizHomogeneaDH(0,90,0,T5)
    H56 = matrizHomogeneaDH(0,0,L6,T6)

    H02 = H01.dot(H12)
    H03 = H02.dot(H23)
    H04 = H03.dot(H34)
    H05 = H04.dot(H45)
    H06 = H05.dot(H56)

    P0 = np.array([[0],[0],[0],[1]])
    P1 = H01[:,3]
    P1f = P1[:,np.newaxis]
    P2 = H02[:,3]
    P2f = P2[:,np.newaxis]
    P3 = H03[:,3]
    P3f = P3[:,np.newaxis]
    P4 = H04[:,3]
    P4f = P4[:,np.newaxis]
    P5 = H05[:,3]
    P5f = P5[:,np.newaxis]
    P6 = H06[:,3]
    P6f = P6[:,np.newaxis]
    R = np.concatenate([P0,P1f,P2f,P3f,P4f,P5f,P6f], axis=1)

    roll = np.arctan2(H06[2,1],H06[2,2])*(180/np.pi)
    pitch = np.arctan2(-H06[2,0],np.sqrt((H06[2,1]**2)+(H06[2,2]**2)))*(180/np.pi)
    yaw = np.arctan2(H06[1,0],H06[0,0])*(180/np.pi)
    r = np.array([[roll,pitch,yaw]])

    return R, r
```

Fig. 5 Cinemática directa en Python.

2.5 CINEMÁTICA INVERSA.

El desacoplo cinemático consiste en dividir el sistema en dos problemas (posición y orientación). Para ello, dada una posición y orientación final deseadas, establece las coordenadas del punto de corte de los 3 últimos ejes (muñeca del robot) calculándose los valores de las tres primeras variables articulares $(\theta_1, \theta_2, \theta_3)$ que consiguen posicionar este punto. [10] [9]
Se realiza el desacoplo cinemático restando la posición del robot con tamaño de la muñeca por el vector unitario de la orientación del efector final ($l_4\ \vec{a}_6$)

$$\vec{p}_3 = \vec{p}_6 - l_4\ \vec{a}_6$$

La función para la cinemática inversa del robot de seis grados de libertad consiste en que, a partir de la generación de la matriz de transformación homogénea del último eslabón, se

sigan los siguientes pasos para la obtención de los ángulos articulares (ver fig. 6, 7 y 8):
1. Calcular el centro de la muñeca con la posición y orientación deseadas.
2. Obtener las posiciones de T1, T2 y T3.
3. Generar la matriz de rotación del tercer eslabón con los parámetros de Denavit-Hartenberg.
4. Obtener la matriz de rotación R36.
5. Obtener los ángulos T4, T5 y T6

```python
import numpy as np
from matrizHomogeneaRotacion import degToRad
from matrizHomogeneaDH import *

def robot6Gd1CI(X, Y, Z, L1, L2, L3, L4, L5, L6, codo, muneca, Roll, Pitch, Yaw):
    matrizRotacion = eulerRPY(Roll, Pitch, Yaw)
    matrizPosicion = np.array([[X],[Y],[Z]])
    H06p1 = np.append(matrizRotacion, matrizPosicion, axis = 1)
    H06p2 = np.array([0, 0, 0, 1])
    H06 = np.append(H06p1, [H06p2] , axis = 0)

    """1) Calcular el centro de la muneca con la posicion y orientacion deseadas. """
    ox = H06[0,3]
    oy = H06[1,3]
    oz = H06[2,3]

    r13 = H06[0,2]
    r23 = H06[1,2]
    r33 = H06[2,2]
    r12 = H06[0,1]
    r22 = H06[1,1]
    r32 = H06[2,1]
    r11 = H06[0,0]
    r21 = H06[1,0]
    r31 = H06[2,0]

    xc = ox - L6*r13
    yc = oy - L6*r23
    zc = oz - L6*r33

    """2) Obtener las posiciones de T1, T2 y T3. """
    T1 = np.arctan2(yc, xc)
    zc = zc - L1
    D = ((xc**2)+(yc**2)+(zc**2)-(L2**2)-(L4**2))/(2*L2*L4)
    T3 = np.arctan2(codo*np.sqrt(1-(D**2)),D)
    k1 = L2+(L4*np.cos(T3))
    k2 = L4*np.sin(T3)
    T2 = np.arctan2(zc,np.sqrt((xc**2)+(yc**2)))-np.arctan2(k2,k1)
    T3 = T3 + (np.pi/2)
```

Fig. 6 Cinemática inversa en Python, parte 1.

```python
"""3) Obtener la matriz de rotacion H03 con los parametros DH. """
T1 = T1*(180/np.pi)
T2 = T2*(180/np.pi)
T3 = T3*(180/np.pi)

H01 = matrizHomogeneaDH(0,90,L1,T1)
H12 = matrizHomogeneaDH(L2,0,0,T2)
H23 = matrizHomogeneaDH(L3,90,0,T3)
H02 = H01.dot(H12)
H03 = H02.dot(H23)
print(H03)

"""4) Obtener la matriz de rotacion R36. """
R06 = H06[0:3,0:3]
R03 = H03[0:3,0:3]
print(R03)
invR03 = np.linalg.inv(R03)
R36 = invR03.dot(R06)

ax = R36[0,2]
ay = R36[1,2]
az = R36[2,2]

sx = R36[0,1]
sy = R36[1,1]
sz = R36[2,1]

nx = R36[0,0]
ny = R36[1,0]
nz = R36[2,0]
```

Fig. 7 Cinemática inversa en Python, parte 2.

```python
"""5) Obtener los angulos T4, T5, T6. """

if muneca == -1:
    T4 = np.arctan2(ay,ax)*(180/np.pi)
    T5 = np.arctan2(np.sqrt((ax**2)+(ay**2)),az)*(180/np.pi)
    T6 = np.arctan2(sz,-nz)*(180/np.pi)
else:
    T4 = np.arctan2(-ay,-ax)*(180/np.pi)
    T5 = np.arctan2(-np.sqrt((ax**2)+(ay**2)),az)*(180/np.pi)
    T6 = np.arctan2(-sz,nz)*(180/np.pi)

angulos = [T1, T2, T3, T4, T5, T6]
T12 = T1
T22 = T2
T32 = T3
T42 = T4
T52 = T5
T62 = T6

angulos2 = [T12, T22, T32, T42, T52, T62]
return angulos, angulos2
```

Fig. 8 Cinemática inversa en Python, parte 3.

3 DISEÑO EN SOLIDWORKS

Se utilizó el software de SolidWorks para el diseño del Robot, se hicieron las piezas de las articulaciones con la finalizar de crear el ensamblaje completo del Robot y de esta manera se pueda crear una simulación del diseño mecánico usando las herramientas del SolidWorks para posteriormente exportarlos en el formato WRL para manipularlos desde LabVIEW.

En la tabla 1 se puede observar los diseños que se crearon.

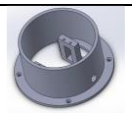
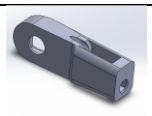
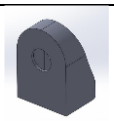
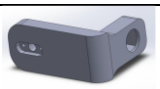
Nombre	Pieza	Descripción
Base		Base fija del robot
Waist		Base giratoria que representa el movimiento de la cadera del robot
Arm1		Primer eslabón que re presenta el movimiento del hombro del robot
Arm2		Segundo eslabón que representa el movimiento del codo del robot.
Arm3		Pieza giratoria que representa el movimiento del antebrazo del robot
Muñeca		Pieza que representa el movimiento de la muñeca

Tabla 1.- piezas del robot en SolidWorks

4 SIMULADOR EN LabVIEW.

El software LabVIEW tiene como ventaja la versatilidad de integración con otros softwares y de la misma manera es muy fácil la implementación con hardware usando tarjetas de adquisición de datos o microcontroladores. Por esta razón se utilizó este programa para desarrollar el simulador del robot y posteriormente unirlo con el prototipo físico.

4.1 Graficar en LabVIEW el robot en 3D

El diagrama de bloques del sub VI se ocupa de la animación de rotación de la representación en 3D del robot de seis grados de libertad, según los datos recibidos de la VI principal (Fig. 9).

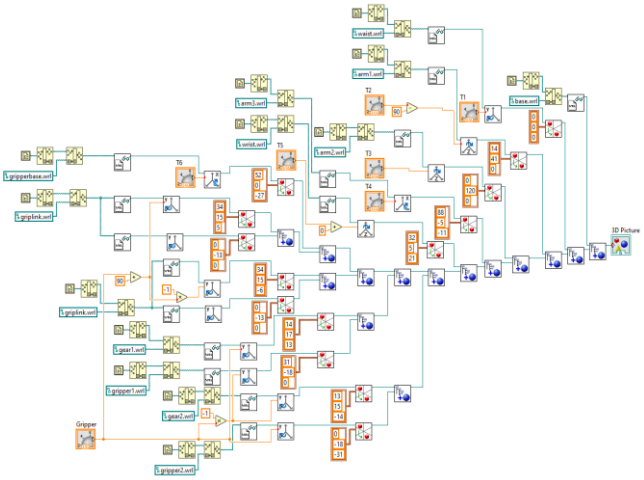


Fig.9 Diagrama a bloques de la sub VI de Show 6DOF robot.

El apartado gráfico se dedica a mostrar de forma digital la posición actual, trayectoria y límites de movimiento del elemento robótico con el modelo 3D (fig. 10)

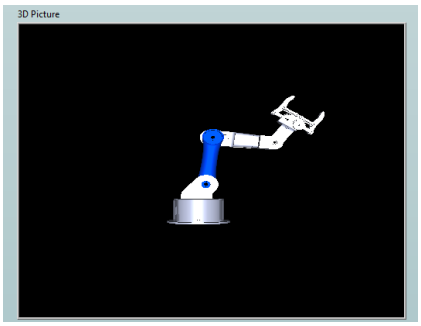


Fig. 10 Modelo en 3D del robot de seis grados de libertad.

4.2 Robot 6GDL Cinemática Directa.

En este sub VI se utiliza Python Node para llamar el *script* de cinemática directa, empleado en el cálculo de la solución de la cinemática directa a fin de obtener la posición y orientación final (fig. 11). en la fig. 12 muestra el resultado grafico de esta SubVI

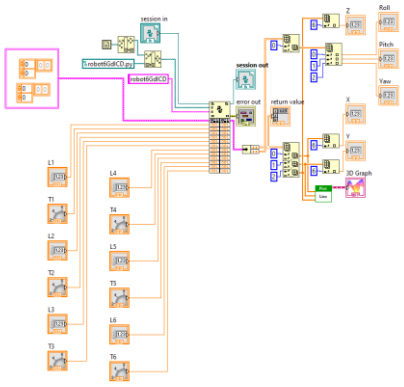


Fig.11 Diagrama a bloques de la sub VI de Robot 6GDL CD.

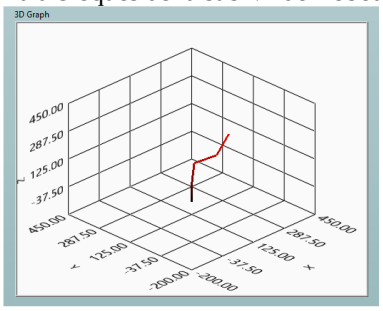


Fig.12 Representación en 3D del robot de seis grados de libertad.

4.3 Robot 6GDL Cinemática Inversa.

En la sub VI de *Robot 6GDL CI* hace uso de igual forma de Python Node, pero para llamar al *script* de cinemática inversa, con la finalidad de conseguir el resultado de los ángulos de las articulaciones (fig. 13).

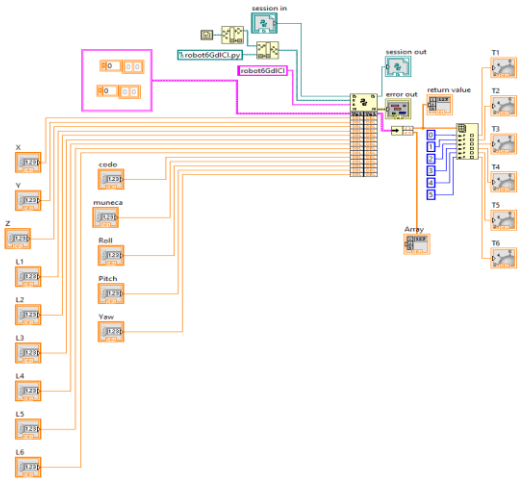


Fig. 13 Diagrama a bloques de la sub VI de Robot 6GDL CI.

4.4 Calcular pasos.

La sub VI de *Calcular pasos* simplemente hace el cálculo de los incrementos que se harán para llevar a cabo la animación con las variables de posición (X, Y y Z) y de orientación (Roll, Pitch y Yaw), tomando en cuenta el valor actual, el valor nuevo y cuantos pasos se requieren hacer para llegar a ese nuevo valor (fig. 14).

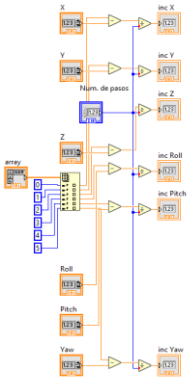


Fig. 14 Diagrama a bloques de la sub VI de Calcular pasos.

4.5 Paso aumento

El propósito del sub VI es actualizar los valores de las variables de posición (X, Y y Z) y de orientación (Roll, Pitch y Yaw) de acuerdo al valor de incremento que se ha obtenido en la sub VI de *Calcular pasos* (fig. 15).

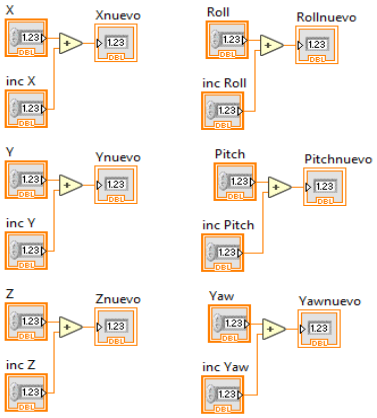


Fig. 15 Diagrama a bloques de la sub VI de Paso aumento.

4.6 Bucles del VI principal.

El bucle de control cumple las siguientes actividades:

- Apartado gráfico: actualiza los gráficos 3D cuando el usuario interactúa con la parte de cinemática directa y/o inversa en el panel frontal.
- Cinemática directa: toman los valores de las perillas del panel frontal y con el uso de la sub VI de cinemática directa se modifican los valores de posición y orientación.
- Cinemática inversa: se guardan los valores introducidos en los indicadores de posición y orientación en el arreglo de *Lista de movimientos*, cuando es presionado el botón *Agregar*. De igual manera, al tener introducidos los valores en el arreglo, la secuencia de movimientos es activada al accionar el botón de *Realizar secuencia* y se crea una animación. Esta animación se inicia con bucle *for* que recorre el arreglo de *Lista de movimientos*, en el cual dentro de sí existe un bucle *while* que calcula los pasos a realizar en la animación con la sub VI de *Calcular pasos* y, por último, otro bucle *while* (Fig.16a y 16b) que se ubica en el interior del anterior bucle, que es responsable de actualizar/ aumentar el valor de los elementos usados en la animación con la sub VI *Paso aumento* y así ir haciendo la animación con las sub VIs de cinemática directa e inversa, necesarios para la solución de la cinemática inversa completa. Asimismo, este último actualiza el apartado gráfico, los valores de los ángulos de las articulaciones, de posición y orientación en el panel frontal.
- Agregar elementos a la cola: Cuando el botón de realizar secuencia no es presionado, se agregan valores a la cola cuando se cambian los valores de las perillas del panel frontal. Igualmente, se agregan valores a la cola en el *while* (Fig.16a y 16b) *que* lleva a cabo los cálculos de la cinemática inversa.

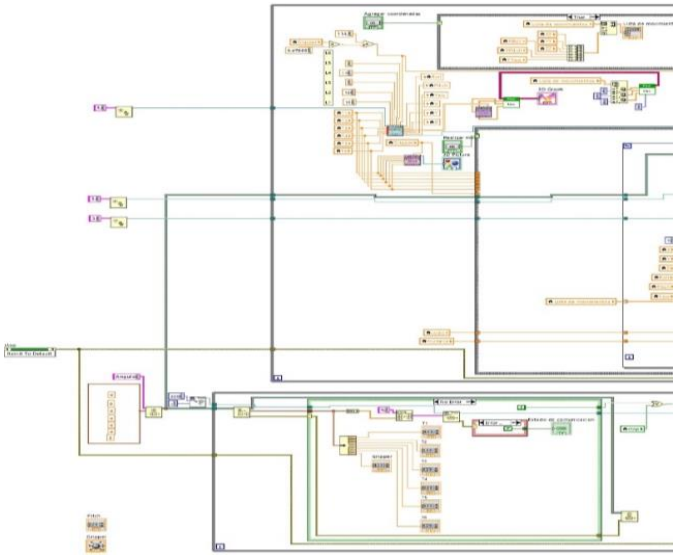


Fig. 16a Diagrama a bloques del bucle *while* de control.

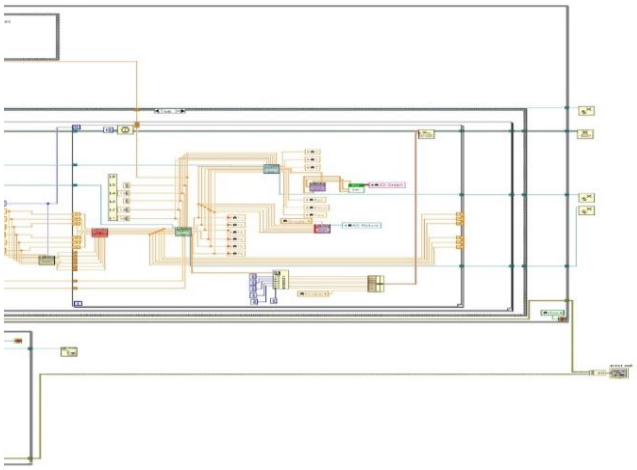


Fig. 16b Diagrama a bloques del bucle *while* de control.

5 RESULTADOS

Como resultado se pudo obtener un simulador de 6GDL en LabVIEW aplicando la cinemática con el lenguaje Python, en las pruebas realizadas se pudo observar que el simulador quedo muy estable a diferencias de otros trabajos realizados, en la figura 17 se muestra la interfaz del robot. Con la interfaz el usuario puede interactuar con el simulador del Robot con los controladores manipulando el robot con los movimientos por articulación Joint, así mismo también se puede mover el robot por World y Tool. Se pueden guardar las posiciones para hacer un recorrido entre ellas, simulando una trayectoria.

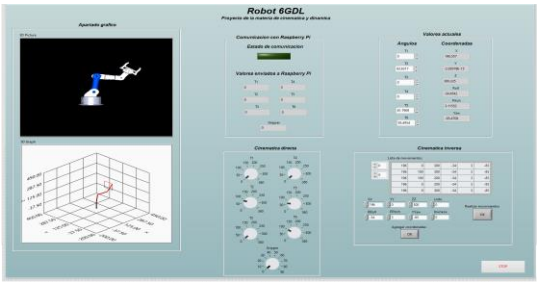


Fig. 17 Panel frontal de LabVIEW completo.

6 CONCLUSIONES

Con este trabajo se pudo comprobar la aplicación Python en LabVIEW para el desarrollo del robot de 6GDL, en las pruebas realizadas se pudo observar que la aplicación de la matemática en Python es muy estable, anteriormente se desarrolló este robot utilizando el módulo MathScript pero se han tenido muchos problemas con la conexión de los scripts de Matlab. [4].

Como los scrips de los modelados matemáticos están realizados en Python, como trabajo futuro se podrá realizar el control de un prototipo físico del robot directamente desde una RaspBerry Pi

7 REFERENCIAS

[1] J. E. P. Ruiz, Modelación Cinematico del robot CRS A465 Utilizando el algebra de quaterniones, Mexico D.F.: Universidad Nacional Autónoma de México, 2006.

[2] J. S. M. Erazo, Simulación de un Robot con 6 Grados de Libertad Utilizando ToolBox Robotics del Software Matlab, Escuela Politecnica Nacional, 2020.

[3] Ignacio Vázquez Cuevas, «Modelación y Diseño de un Simulador de un Robot Paralelo Manejado por un Controlador Manual Didáctico,» *Congreso Nacional de Mecatrónica,* vol. 8, pp. 329-324, 2009.

[4] V. C. Ignacio, «Simulador de Robot de 6GDL Integrando Matlab,Solidworks Y LalVIEW,» *ELECTRO,* vol. 43, nº Robotica, pp. 180-185, 2021.

[5] J. Vega, «PyBoKids: An Innovative Python-Based Educational Framework Using Real and Simulated,» *ELECTRONICS,* nº Arduino Robots, pp. 1-16, 2019.

[6] M. M. Jorge, «Simulación Robotica Colaborativa con ROS,» Univercidad de Valladolid Escual de Ingenierias Industriales, Valladolid, 2020.

[7] LorisFichera, «A Python framework for programming autonomous robots using a declarative approach,» *Science of Computer Programming,* vol. 139, nº Robot programming, pp. 36-55, 2017.

[8] R. Poncelas Bodelón, «Desarrollo del software de control de un brazo robotizado mediante Python,» Univercidad de Valladolid Escual de Ingenierias Industriales, Valladolid, 2014.

[9] A. Barrientos, Fundamentos de Robótica, Madrid España: McGraw- HILL, 1997.

[10] J. J. Graic, Robótica, Mexico DF: PEARSON, 2006.

[11] F. R. CORTÉS, Matlab Aplicado a Robotica y Mecatronica, MEXICO DF: ALFAOMEGA, 2012.

[12] W. S. B. Escobedo, «Modelado y simulación de un manipulador de seis grados de libertad para aplicaciones industriales,» *Pueblo Cont.*, vol. 24, nº 1, pp. 15-27, 2013.

[13] A. R. Ramos, Simulacion de un Brazo Robotico en Simmechanics, Madrid: Universidad Carlos III de madrid , 2015.

[14] D. Chaos, «Virtual and Remote Robotic Laboratory Using EJS, MATLAB and LabVIEW,» *SENSORS,* vol. 13, nº Virtual and Remote Robotic, pp. 2595-2612, 2013.