Trabajo practico Nro. 1

Objetivo:

Analizar la cinemática del robot M5 por medio de MATLAB para el estudio de su movimiento con respecto a un sistema de referencia.
Implementar en lenguaje C del CodeWarrior para el micro DSP56800/E. Y verificar la implementación.

Introducción sobre la cinemática del robot:

Introducción teórica:

La mecánica es la parte de la física que estudia el movimiento, esta se subdivide en:

- ***Estática:*** Estudia las fuerzas en equilibrio mecánico.
- ***Cinemática:*** Estudia el movimiento sin tener en cuenta las causas que lo producen.
- ***Dinámica:*** Estudia los movimientos y las consecuencias que lo producen.

De estas tres formas de estudiar el movimiento en el presente trabajo práctico lo haremos a través de la cinemática, de la cual se introducen a continuación los conceptos teóricos.

Cinemática:

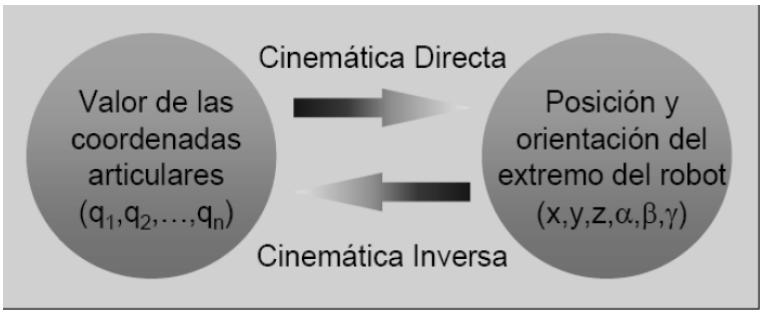
La cinemática es la parte de la mecánica que estudia las leyes del movimiento de los cuerpos sin tener en cuenta las causas que lo producen, limitándose al estudio de la trayectoria en función del tiempo.
La cinemática se subdivide en:

- ***Cinemática Inversa:*** Determina la configuración que debe adoptar el robot para alcanzar la posición y orientación del extremo
- ***Modelo Diferencial:*** Relaciona las velocidades de las articulaciones y las del

extremo del robot.

- ***Cinemática Directa:*** Determina la posición y orientación del extremo final del robot respecto al sistema de coordenadas de referencia, una vez conocidos los ángulos de las articulaciones y los parámetros geométricos de los elementos del robot.

Entre la cinemática directa e inversa podemos fijar la siguiente relación



Modelos cinemáticas

- *Método basado en relaciones geométricas*

  No es sistemático

  Es válido para robots de pocos grados de libertad

- *Método basado en matrices de transformación homogéneas*

  Es sistemático

  Es válido para robots con muchos grados de libertad

Para el desarrollo del estudio del movimiento del ROBOT M5 escogimos hacerlo por cinemática directa, dicho método tiene las ventajas de: vale para robots de muchos grados de libertad, es sistemático y se basa en matrices de transformación homogéneas. Estos tres ítems serán descriptos a continuación.

Cinemática directa por matriz homogénea

Por medio de este método el problema cinemática directo se reduce a encontrar una matriz homogénea de transformación "T" que relacione la posición y orientación del extremo del robot respecto del sistema de referencias situado en la base del mismo. Esta matriz "T" será función de las coordenadas articulares.

Un robot de n grados de libertad está formado por n eslabones unidos por n articulaciones, de forma que cada par articulación-eslabón constituye un grado de libertad. De esta forma podemos notar que el M5 tiene 5 grados de libertad.

A cada eslabón se le puede asociar un sistema de referencias solidario a él y, utilizando las transformaciones homogéneas, es posible representar las rotaciones y traslaciones relativas entre los distintos eslabones que componen al robot.

La matriz que representa la posición y orientación relativa entre los sistemas asociados a dos eslabones consecutivos del robot suele denominarse matriz ${}^{i-1}A_{i,}$

Así, en nuestro M5 la posición y orientación del sistema del quinto eslabón del robot respecto a la base del mismo donde se encuentra el sistema de coordenadas de referencias pude verse como:

$$ {}^{0}A_5 = {}^{0}A_1 \, . \, {}^{1}A_2 \, . \, {}^{2}A_3 \, . \, {}^{3}A_4 \, . {}^{4}A_5 $$

Cundo se consideran todos los grados de libertad mediante cada una de las matrices, la matriz resultante se la denomina matriz de transformación T.

Para describir la relación que existe entre dos elementos contiguos hicimos uso de la representación de Denavit-Hanternber (D-H), este método matricial permite establecer de manera sistemática un sistema de coordenadas ligado a cada eslabón de una cadena articulada, pudiéndose determinar a continuación las ecuaciones cinemáticas de la cadena completa.

Según D-H, escogiendo adecuadamente las coordenadas asociadas a cada eslabón, será posible pasar de un eslabón al siguiente mediante 4 transformaciones básicas que dependen exclusivamente de las características geométricas de cada eslabón.

Estas 4 transformaciones básicas permiten relacionar el sistema de referencias del elemento i con el sistema del elemento i-1.

El paso del sistema S_{i-1} al S_i mediante las 4 transformaciones básicas está garantizado solo se los sistemas han sido definidos de acuerdo a las siguientes reglas básicas:

1 – Rotación alrededor del eje z_{i-1} un ángulo θ_i

2 – Traslación a lo largo de Z_{i-1} una distancia d_i ; vector d_i $(0,0,d_i)$

3 - Traslación a lo largo de X_{i-1} una distancia a_i ; vector d_i $(0,0,a_i)$

4 – Rotación alrededor del eje X_{i-1} un ángulo α_i

Donde θ_i - α_i – a_i – d_i son los parámetros D-H del eslabón.

Siguiendo los pasos antes mencionados la matriz queda de la siguiente forma:

$$^{i-1}A_i = T(Z, \theta_i) \cdot T(0,0,d_i) \cdot T(a_i,0,0) \cdot T(X, \alpha_i) =$$

$$= \begin{vmatrix} C\theta_i & -S\theta_i & 0 & 0 \\ S\theta_i & C\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix} \cdot \begin{vmatrix} 1 & 0 & 0 & 0 \\ 0 & C\alpha_i & -S\alpha_i & 0 \\ 0 & S\alpha_i & C\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

$$\begin{vmatrix} C\theta_i & -C\alpha_i S\theta_i & S\alpha_i S\theta_i & a_i C\theta_i \\ S\theta_i & C\alpha_i C\theta_i & -S\alpha_i C\theta_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{vmatrix}$$

La matriz definida anteriormente junto con la definición de los 4 parámetros de D-H conforman el siguiente algoritmo para la resolución del problema cinemática directo.

A continuación definiremos el Algoritmo de D-H para la obtención del modelo cinemática directo.

Algoritmo de Denavit-Hartenberg

D-H 1. Numerar los eslabones comenzando con 1 (primer eslabón móvil de la cadena) y acabando con n (último eslabón móvil). Se numerara como eslabón 0 a la base fija del robot.

D-H 2. Numerar cada articulación comenzando por 1 (la correspondiente al primer grado de libertad) y acabando en n.

D-H 3. Localizar el eje de cada articulación. Si esta es rotativa, el eje será su propio eje de giro. Si es prismática, será el eje a lo largo del cual se produce el desplazamiento.

D-H 4. Para i de 0 *a* n-1 situar el eje **z**i sobre el eje de la articulación i + 1.

D-H 5. Situar el origen del sistema de la base {S0} en cualquier punto del eje **z**0 .Los ejes **x**0 e **y**0 se situaran de modo que formen un sistema dextrógiro con **z**0.

D-H 6. Para i de 1 a n-1, situar el sistema { Si }(solidario al eslabón i) en la intersección del eje **z**i con la línea normal común a **z**i-1 y **z**i .Si ambos ejes se cortasen se situaría {Si} en el punto de corte. Si fuesen paralelos { Si } se situaría en la articulación i + 1.

D-H 7. Situar **x**i en la línea normal común a **z**i-1 y **z**i .

D-H 8. Situar **y**i de modo que forme un sistema dextrógiro con **x**i y **z**i.

D-H 9. Situar el sistema { Sn } en el extremo del robot de modo que **z**n , coincida con la dirección de **z**n-1 y **x**n sea normal a **z**n-1 y **z**n.

D-H 10. Obtener θi como el ángulo que hay que girar en torno a **z**i-1 para que **x**i-1 y **x**i , queden paralelos.

D-H 11. Obtener di, como la distancia, medida a lo largo de **z**i-1,que habría que desplazar { Si-1 } para que **x**i y **x**i-1 quedasen alineados.

DH 12. Obtener ai como la distancia medida a lo largo de **x**i (que ahora coincidiría con **x**i-1)que habría que desplazar el nuevo { Si-1 } para que su origen coincidiese con { Si }.

DH 13. Obtener αi como el ángulo que habría que girar entorno a **x**i, (que ahora coincidiría con **x**i-1),para que el nuevo { Si-1 } coincidiese totalmente con { Si }.

DH 14. Obtener las matrices de transformación i-1Ai definidas anteriormente.

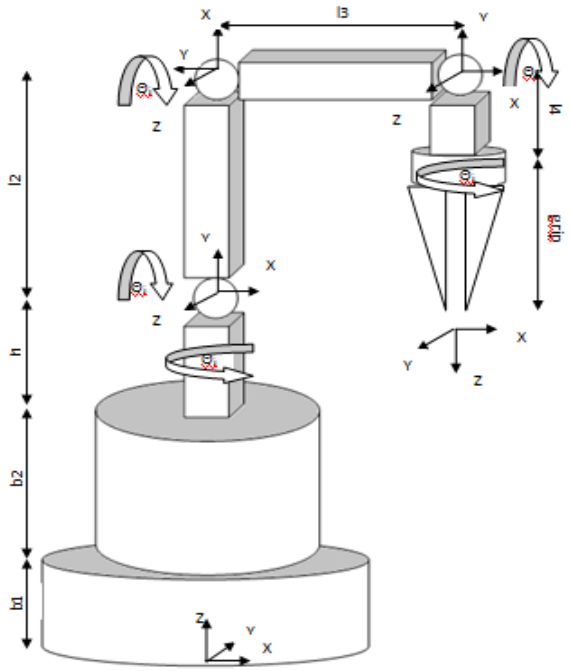
DH 15. Obtener la matriz de transformación que relaciona el sistema de la base con el del extremo del robot **T** = 0A1, 1A2 .....n-1An

DH 16. La matriz **T** define la orientación (submatriz de rotación) y posición (submatriz de traslación) del extremo referido a la base en función de las **n** coordenadas articulares.

Los cuatro parámetros de D-H (θ_i - α_i – a_i – d_i):

- θ_i - Es el ángulo que forman los ejes X_{i-1} y X_i, medido en un plano perpendicular al eje Z_{i-1}, utilizando la regla de la mano derecha. Se trata de un parámetro variable en articulaciones giratorias.
- d_i - Es la distancia a lo largo del eje Z_{i-1}. Es un parámetro variable en articulaciones prismáticas.
- a_i - Es la distancia a los largo del eje X_i. Es un parámetro variable en articulaciones giratorias.
- α_i - Es el ángulo de separación del eje Z_{i-1} y el eje Z_i utilizando la regla de la mano derecha.

Modelo cinemática directo del M-5:



ARTICULACION	θ	d	a	α
1	q1	b1+b2+l1	0	90
2	q2-90	0	l2	0
3	q3-90	0	l3	0
4	q4-90	0	l4	0
5	-90	0	0	90
6	q5	grip	0	0

Matriz homogénea de la cinemática directo del M-5 en Matlab:

```
syms b1 b2 l1 l2 l3 l4 q1 q2 q3 q4 q5 grip;
syms x0 y0 z0;
pi1=sym('pi');
A01=MDH(q1,b1+b2+l1,0,pi1/2);
A12=MDH(q2-(pi1)/2,0,l2,0);
A23=MDH(q3-(pi1)/2,0,l3,0);
A34=MDH(q4-(pi1)/2,0,l4,0);
A45=MDH(-(pi1)/2,0,0,pi1/2);
A56=MDH(q5,grip,0,0);
A46=A45*A56;
T = A01*A12*A23*A34*A46;
xyz=simplify(T*[x0;y0;z0;1]);
A=simple(xyz);

x=A(1);y=A(2); z=A(3);
```

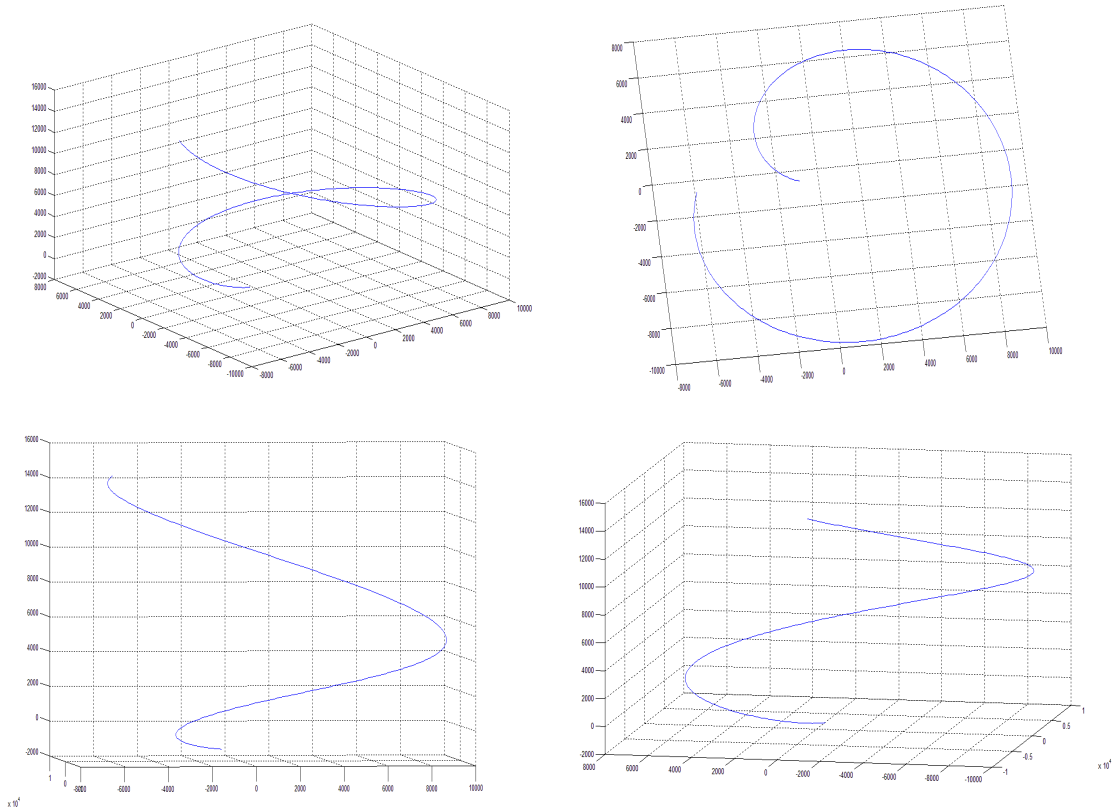
Desarrollo e implementación en CodeWarriror DSP56800/E

```
/* MODULE TPN1 */
#include "Cpu.h"
#include "Events.h"
#include "TFR1.h"
#include "MFR1.h"
#include "MEM1.h"
#include "XFR1.h"
#include "AFR1.h"
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "stdio.h"

//*****************************************************************************
//              DEFINE
//*****************************************************************************
#define MXRAD 361 //100
#define PULSE2RAD 32767/MXRAD   // 32767/100 impulsos // #define PULSE2RAD 450
//*****************************************************************************
//              GLOBAL VARIABLE
//*****************************************************************************
Frac16 Cos1,Cos2,Cos3,Cos4,Cos5,Sin1,Sin2,Sin3,Sin4,Sin5,CosA,CosB,CosC,SinB,SinC;
Frac16 b1,b2,l1,l2,l3,l4,q1,q2,q3,q4,q5,grip;
Frac16 x, y, z, x0, y0, z0,ZA,ZB,ZC;
int i,j,k;

void main(void)
{
  /* Write your local variable definition here */
  /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
  /*** End of Processor Expert internal initialization.                    ***/

  /* Write your code here */
        b1=(0x10000/100)*10;b2=0x10000*5/100;
        l1=(0x10000/100)*10;l2=0x10000*10/100;l3=0x10000*10/100;l4=0x10000*10/100;grip=0x10000*2/100;
        x0=(0x10000/100)*1;y0=(0x10000/100)*1;z0=(0x10000/100)*1;
        printf ("₩nx₩ty₩tz");
        printf ("₩n₩n!!!!!!!!!! VARIANDO q4<q3<q1 y q5=q1 !!!!!!!!!!!!₩n₩n");

        for(q1=0,q2=0,q3=0,q4=0,q5=0;(q1<0xFFFF-200);q1 += 200,q3 += 50,q4 += 50,q5 += 200)
        {
                ZA=add(q2,add(q3,q4));
                ZB=add(ZA,q5);
                ZC=sub(ZA,q5);
```

```
                Cos1 = tfr16CosPIx (q1);
                Cos2 = tfr16CosPIx (q2);
                Cos3 = tfr16CosPIx (q3);
                Cos4 = tfr16CosPIx (q4);
                Cos5 = tfr16CosPIx (q5);
                Sin1 = tfr16SinPIx (q1);
                Sin2 = tfr16SinPIx (q2);
                Sin3 = tfr16SinPIx (q3);
                Sin4 = tfr16SinPIx (q4);
                Sin5 = tfr16SinPIx (q5);
                CosA = tfr16CosPIx (ZA);
                CosB = tfr16CosPIx (ZB);
                CosC = tfr16CosPIx (ZC);
                SinB = tfr16SinPIx (ZB);
                SinC = tfr16SinPIx (ZC);

                x= add(add(mult(l2,mult(Cos1,Sin2)),add(mult(y0,mult(Cos5,Sin1)),mult(x0,mult(Sin1,Sin5)))),
add(add(negate(mult(l3,mult(Cos1,mult(Cos2,Cos3)))),mult(l3,mult(Cos1,mult(Sin2,Sin3)))),
add(add(mult(grip,mult(mult(Cos1,Cos2),mult(Cos3,Sin4))),mult(grip,mult(mult(Cos1,Cos2),mult(Cos4,Sin3)))),
add(add(mult(grip,mult(mult(Cos1,Cos3),mult(Cos4,Sin2))),negate(mult(l4,mult(mult(Cos1,Cos2),mult(Cos3,Sin4))))),
add(negate(add(mult(l4,mult(mult(Cos1,Cos2),mult(Cos4,Sin3))),mult(l4,mult(mult(Cos1,Cos3),mult(Cos4,Sin2))))),
add(add(mult(z0,mult(mult(Cos1,Cos2),mult(Cos3,Sin4))),mult(z0,mult(mult(Cos1,Cos2),mult(Cos4,Sin3)))),
add(add(mult(z0,mult(mult(Cos1,Cos3),mult(Cos4,Sin2))),negate(mult(grip,mult(mult(Cos1,Sin2),mult(Sin3,Sin4))))),
add(add(mult(l4,mult(mult(Cos1,Sin2),mult(Sin3,Sin4))),negate(mult(z0,mult(mult(Cos1,Sin2),mult(Sin3,Sin4))))),
add(negate(add(mult(mult(y0,Cos1),mult(mult(Cos2,Cos3),mult(Cos4,Sin5))),mult(mult(x0,Cos1),mult(mult(Cos2,Cos5),mult(Sin3,Sin4))))),
add(negate(add(mult(x0,mult(Cos1,mult(mult(Cos3,Cos5),mult(Sin2,Sin4)))),mult(x0,mult(Cos1,mult(mult(Cos4,Cos5),mult(Sin2,Sin3)))))),
add(add(mult(y0,mult(Cos1,mult(mult(Cos2,Sin3),mult(Sin4,Sin5)))),mult(y0,mult(Cos1,mult(mult(Cos3,Sin2),mult(Sin4,Sin5))))),
add(mult(y0,mult(Cos1,mult(mult(Cos4,Sin2),mult(Sin3,Sin5)))),mult(x0,mult(Cos1,mult(mult(Cos2,Cos3),mult(Cos4,Cos5)))))))))))))));

                y=add(add(mult(l2,mult(Sin1,Sin2)),negate(add(mult(x0,mult(Cos1,Sin5)),mult(y0,mult(Cos1,Cos5))))),
add(add(negate(mult(l3,mult(Cos2,mult(Cos3,Sin1)))),mult(l3,mult(Sin1,mult(Sin2,Sin3)))),
add(add(mult(grip,mult(mult(Cos2,Cos3),mult(Sin1,Sin4))),mult(grip,mult(mult(Cos2,Cos4),mult(Sin1,Sin3)))),
add(add(mult(grip,mult(mult(Cos3,Cos4),mult(Sin1,Sin2))),negate(mult(l4,mult(mult(Cos2,Cos3),mult(Sin1,Sin4))))),
add(negate(add(mult(l4,mult(mult(Cos2,Cos4),mult(Sin1,Sin3))),mult(l4,mult(mult(Cos3,Cos4),mult(Sin1,Sin2))))),
add(add(mult(z0,mult(mult(Cos2,Cos3),mult(Sin1,Sin4))),mult(z0,mult(mult(Cos2,Cos4),mult(Sin1,Sin3)))),
add(add(mult(z0,mult(mult(Sin3,Sin4),mult(Sin1,Sin2))),negate(mult(grip,mult(mult(Sin1,Sin2),mult(Sin3,Sin4))))),
add(add(mult(l4,mult(mult(Sin1,Sin2),mult(Sin3,Sin4))),negate(mult(z0,mult(mult(Sin1,Sin2),mult(Sin3,Sin4))))),
add(add(mult(x0,mult(Cos2,mult(mult(Cos3,Cos4),mult(Cos5,Sin1)))),negate(mult(y0,mult(Cos2,mult(mult(Cos3,Cos4),mult(Sin1,Sin5)))))),
add(negate(add(mult(mult(x0,Cos2),mult(mult(Cos5,Sin1),mult(Sin3,Sin4))),mult(mult(x0,Cos3),mult(mult(Cos5,Sin1),mult(Sin2,Sin4))))),
add(add(negate(mult(mult(x0,Cos4),mult(mult(Cos5,Sin1),mult(Sin2,Sin3)))),mult(mult(y0,Cos2),mult(mult(Sin1,Sin3),mult(Sin4,Sin5)))),
add(mult(mult(y0,Cos3),mult(mult(Sin1,Sin2),mult(Sin4,Sin5))),mult(mult(y0,Cos4),mult(mult(Sin1,Sin2),mult(Sin3,Sin5)))))))))))))));

                z=add(add(b1,add(b2,add(l1,negate(add(mult(l3,tfr16SinPIx (add(q2,q3))),mult(l2,Cos2)))))),
add(mult(add(mult(y0,CosB),add(mult(x0,SinB),add(mult(x0,SinC),negate(mult(y0,CosC))))),FRAC16(0.5)),
mult(negate(add(grip,add(negate(l4),z0))),CosA)));

                printf ("₩n%d₩t%d₩t%d;", x, y, z);
        }
}
```

Resultado de la simulación:

xyz=['tabla generada por el codewarrior'];plot3(xyz(:,1,1),xyz(:,2,1),xyz(:,3,1));grid;



Verificacion de código C en Matlab

```
b1=5.85;
b2=9.6;
l1=7;
l2=12.7;
l3=12.7;
l4=4.4;
grip=10;
x=zeros(30000,1);y=zeros(30000,1);z=zeros(30000,1);
x0=0;y0=0;z0=0;
q1=0;q2=0;q3=0;q4=0;q5=0;

for i=1:30000
q2=q2+50;
ZA=q2+q3+q4;
ZB=ZA+q5;
ZC=ZA-q5;

x(i)= l2*cos(q1)*sin(q2) + y0*cos(q5)*sin(q1) + x0*sin(q1)*sin(q5)  ...
       - l3*cos(q1)*cos(q2)*cos(q3) + l3*cos(q1)*sin(q2)*sin(q3)  ...
       + grip*cos(q1)*cos(q2)*cos(q3)*sin(q4) +
       grip*cos(q1)*cos(q2)*cos(q4)*sin(q3)   ...
       + grip*cos(q1)*cos(q3)*cos(q4)*sin(q2) -
       l4*cos(q1)*cos(q2)*cos(q3)*sin(q4)  ...
```

```
       - l4*cos(q1)*cos(q2)*cos(q4)*sin(q3) -
       l4*cos(q1)*cos(q3)*cos(q4)*sin(q2)  ...
       + z0*cos(q1)*cos(q2)*cos(q3)*sin(q4) +
       z0*cos(q1)*cos(q2)*cos(q4)*sin(q3)  ...
       + z0*cos(q1)*cos(q3)*cos(q4)*sin(q2) -
       grip*cos(q1)*sin(q2)*sin(q3)*sin(q4)  ...
       + l4*cos(q1)*sin(q2)*sin(q3)*sin(q4) -
       z0*cos(q1)*sin(q2)*sin(q3)*sin(q4)  ...
       - y0*cos(q1)*cos(q2)*cos(q3)*cos(q4)*sin(q5) -
       x0*cos(q1)*cos(q2)*cos(q5)*sin(q3)*sin(q4)  ...
       - x0*cos(q1)*cos(q3)*cos(q5)*sin(q2)*sin(q4) -
       x0*cos(q1)*cos(q4)*cos(q5)*sin(q2)*sin(q3)  ...
       + y0*cos(q1)*cos(q2)*sin(q3)*sin(q4)*sin(q5) +
       y0*cos(q1)*cos(q3)*sin(q2)*sin(q4)*sin(q5)  ...
       + y0*cos(q1)*cos(q4)*sin(q2)*sin(q3)*sin(q5) +
       x0*cos(q1)*cos(q2)*cos(q3)*cos(q4)*cos(q5);

y(i)=l2*sin(q1)*sin(q2) - x0*cos(q1)*sin(q5) - y0*cos(q1)*cos(q5)  ...
       - l3*cos(q2)*cos(q3)*sin(q1) + l3*sin(q1)*sin(q2)*sin(q3)  ...
       + grip*cos(q2)*cos(q3)*sin(q1)*sin(q4) +
       grip*cos(q2)*cos(q4)*sin(q1)*sin(q3)  ...
       + grip*cos(q3)*cos(q4)*sin(q1)*sin(q2) -
       l4*cos(q2)*cos(q3)*sin(q1)*sin(q4)  ...
       - l4*cos(q2)*cos(q4)*sin(q1)*sin(q3) -
       l4*cos(q3)*cos(q4)*sin(q1)*sin(q2)  ...
       + z0*cos(q2)*cos(q3)*sin(q1)*sin(q4) +
       z0*cos(q2)*cos(q4)*sin(q1)*sin(q3)  ...
       + z0*cos(q3)*cos(q4)*sin(q1)*sin(q2) -
       grip*sin(q1)*sin(q2)*sin(q3)*sin(q4) ...
       + l4*sin(q1)*sin(q2)*sin(q3)*sin(q4) -
       z0*sin(q1)*sin(q2)*sin(q3)*sin(q4)  ...
       + x0*cos(q2)*cos(q3)*cos(q4)*cos(q5)*sin(q1) -
       y0*cos(q2)*cos(q3)*cos(q4)*sin(q1)*sin(q5)  ...
       - x0*cos(q2)*cos(q5)*sin(q1)*sin(q3)*sin(q4) -
       x0*cos(q3)*cos(q5)*sin(q1)*sin(q2)*sin(q4)  ...
       - x0*cos(q4)*cos(q5)*sin(q1)*sin(q2)*sin(q3) +
       y0*cos(q2)*sin(q1)*sin(q3)*sin(q4)*sin(q5)  ...
       + y0*cos(q3)*sin(q1)*sin(q2)*sin(q4)*sin(q5) +
       y0*cos(q4)*sin(q1)*sin(q2)*sin(q3)*sin(q5) ;

z(i)=b1 + b2 + l1 - l3*sin(q2 + q3) - l2*cos(q2) ...
       + (y0*cos(ZB) + x0*sin(ZB) + x0*sin(ZC) - y0*cos(ZC))*0.5  ...
       -(grip - l4 + z0)*cos(ZA) ;
end
plot3(x,y,z);grid;
```

Conclusiones:

Podemos verificar que por medio del método Denavit-Hartenberg expresada en Matlab y luego codificada para el DSP56800/E en CodeWarrior, se visualiza en Matlab , es acorde a los movimientos que esperábamos tener.

Los problemas que tuvimos además de entender bien el alcance de nuestro objetivo era la codificación que se debía hacer en C para el DSP56800/E de CodeWarrior. Ya que al realizar los cálculos en Matlab (double) tenía que codificar teniendo en cuenta el punto fijo del DSP. Sin que haya desborde en los cálculos y redimensionando las medidas.

Tambien, los problemas que tuvimos que enfrentar son las ecuaciones, que al ser muy largas, cometíamos mucho error. Teníamos a opción de multiplicar matricialmente por medio de una función de CodeWarrior, pero ya habíamos avanzado mucho con el método de resolver la ecuación para cada eje. Por eso optamos resolver de esta manera.

El movimiento que estamos haciendo es, girar la base 360º por medio de la 'q1' y también girar el grid 'q5' que no aporta en los cambios de la posición en el plano xyz. La 'q3' y 'q4' se incrementa con menor paso que la base. Este movimiento nos debería dar la sensación de un espiral que incrementa el diámetro y luego empieza a achicarse.

Este movimiento se verifica por medio del grafico en Matlab.