Universidad Tecnológica Nacional
Facultad Regional Buenos Aires

Trabajo Práctico Nº 1:
Cinemática del robot paralelo delta

Materia: Robótica

Curso: R6051

Año: 2014

Docente: Ing. Hernán Giannetta

ATP: Ing. Damián Granzella, Ing. Lucas Barrera

Alumnos: Agustín G. Gimeno, Gustavo Donnadio

Contenido

Introducción

Los manipuladores paralelos son mecanismos donde los eslabones están conectados entre la referencia y la plataforma al mismo tiempo, produciendo una cadena de eslabones cerrada. Este tipo de manipuladores presenta una complejidad matemática superior a los robots de cadena abierta dado que no existe un método sistemático para la resolución de las ecuaciones que gobiernan su comportamiento.

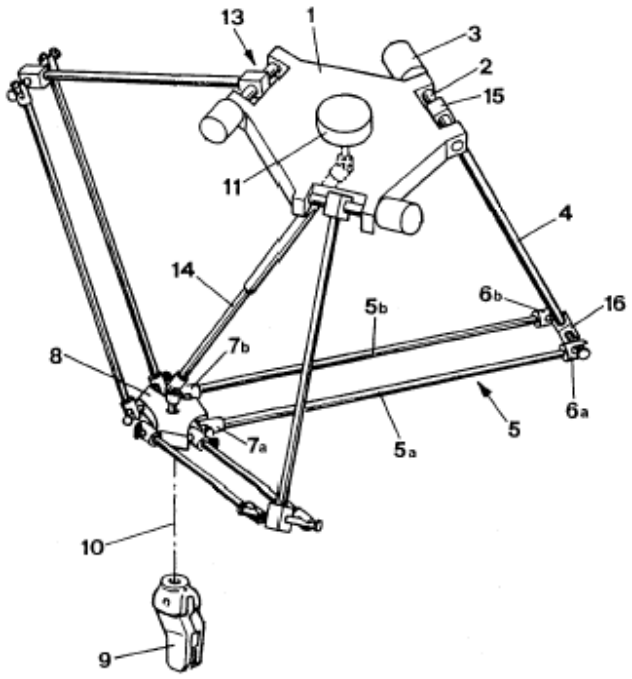
El presente trabajo tiene por fin principal demostrar el procedimiento algebraico realizado para obtener las ecuaciones cinemáticas del robot delta, apoyados por un script realizado en Matlab.

Descripción del robot elegido

Un **robot delta** es un tipo de robot de brazos paralelos. Costa de tres brazos conectados a juntas universales en su base y su característica principal es el uso de paralelogramos en los brazos, que mantienen la orientación del actuador del extremo. Por su velocidad y precisión, los robots delta son muy utilizados en aplicaciones de envasado en fábricas donde se requiere la manipulación de gran cantidad de pequeños productos.

El robot delta fue inventado en los años ochenta por el equipo de investigación dirigido por el catedrático de la Escuela Politécnica Federal de Lausana (EPFL, Suiza), Raymond Clavel.[1] El objetivo de este nuevo tipo de robot era manipular objetos pequeños y ligeros a gran velocidad, que era una necesidad industrial de la época. En 1987, la empresa suiza Demaurex compró la licencia del robot delta y comenzó su producción para la industria del envasado. En 1991, el profesor Reymond Clavel presentó su tesis doctoral titulada: "Concepción de un robot paralelo

rápido de cuatro grados de libertad", y en 1999 recibió en premio del robot de oro. También en 1999, la empresa ABB Flexible Automation comenzó la venta de su robot delta con el nombre comercial de Flexpicker.

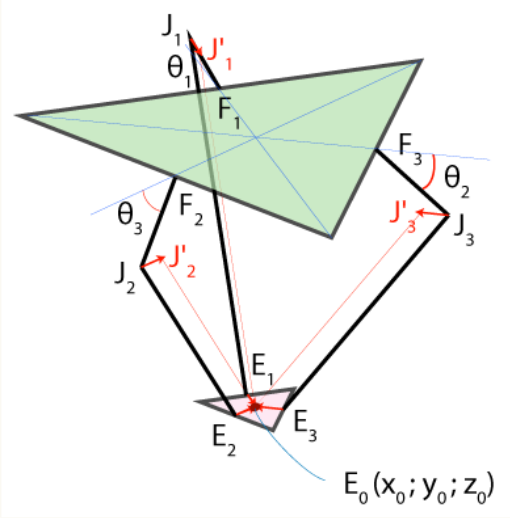


El robot delta consiste de dos plataformas, una superior (1) que es fija a la cual están asociados los tres motores; la segunda plataforma (8) es la punta del robot donde va colocado un actuador (9). Las plataformas están vinculadas entre sí por tres brazos con paralelogramos, los paralelogramos restringen la posición de la plataforma inferior provocando que la misma siempre se encuentre paralela a la plataforma superior. En el caso del diagrama de la figura los motores (3) determinan variando el ángulo de los brazos la posición del cabezal y luego existe un motor adicional (11) que provoca la rotación del actuador en el cabezal.

La principal ventaja de estos robots es la velocidad, comparado con otros tienen la ventaja de tener todos sus actuadores solidarios a su base, teniendo que solo soportar la pieza a mover y sus propios brazos.

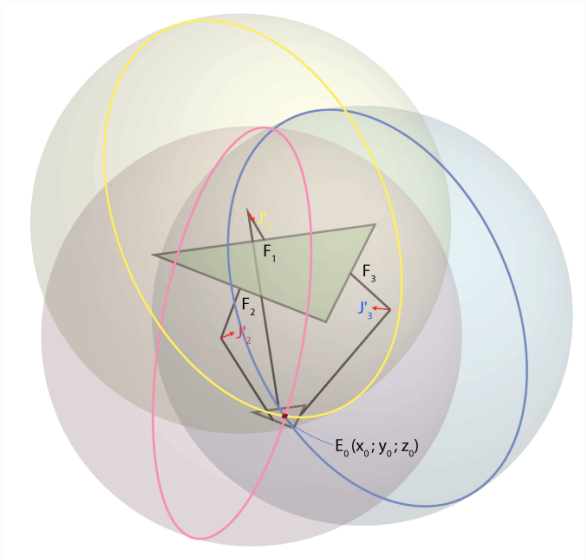
Cinemática directa

La cinemática directa tiene por objetivo conocer la posición de la plataforma inferior, es decir la punta de nuestro robot, en base al ángulo que se le imprimen a los actuadores.



Observando el diagrama anterior, podemos ver que conociendo el largo del brazo sobre el cual actúa directamente el motor y conociendo el ángulo que este le imprime podemos obtener la posición de articulación Ji (siendo i=1,2,3).

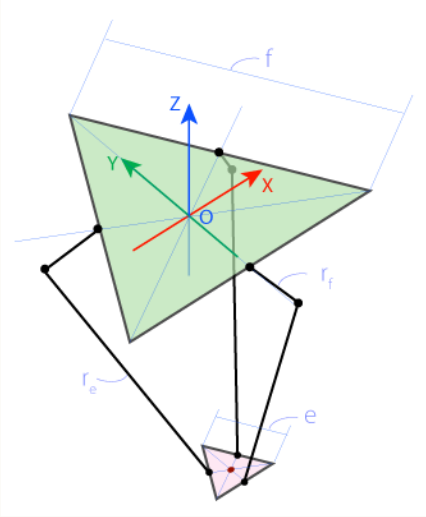
Luego los brazos J1E1, J2E2, J3E3 pueden rotar libremente sobre la articulación Ji correspondiente, formando tres esferas de radio re, como se muestran en la siguiente figura.

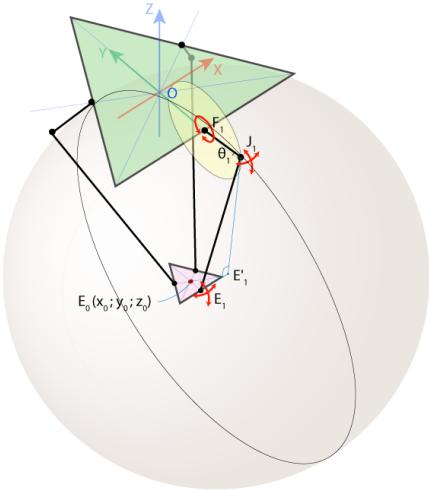


Debemos tener en cuenta además las siguientes consideraciones:

- Las dos plataformas son triángulos equiláteros, aunque de distinto tamaño. Para la nomenclatura subsiguiente se entiende que el lado del triángulo equilátero superior es “f” y el otro es “e”.
- Cada motor está colocado equidistantemente entre sí y del centro. Además se encuentran rotados 120° uno de otros.

- Lo anterior conlleva a que los brazos, rf que son F1J1, F2J2 y F3J3 se mueven con una trayectoria del tipo circular contenida en un plano transversal al eje de los actuadores y por lo tanto también cada uno a 120° del otro.





Por lo tanto para obtener la posición del punto E0(Theta1, Theta2, Theta3) deberemos resolver un sistema de 3 ecuaciones que representan a una esfera de radio re desplazada cada una a una posición Ji(xi,yi,zi), como el que se plantea a continuación:

$$\begin{cases} x^2 + (y - y_1)^2 + (z - z_1)^2 = r_e^2 \\ (x - x_2)^2 + (y - y_2)^2 + (z - z_2)^2 = r_e^2 \\ (x - x_3)^2 + (y - y_3)^2 + (z - z_3)^2 = r_e^2 \end{cases}$$

Para esto se implementó el siguiente script en matlab con el objetivo de demostrar el procedimiento algebraico, algo que en ninguna bibliografía se encontró que llevara su demostración hasta el final. Dicho procedimiento consumió gran parte del tiempo abocado a la realización de este trabajo práctico. Lo interesante es que al observar la salida obtenemos una expresión para (x,y,z) en función de cada una de las variables.

```
syms x y z r
syms x1 x2 x3
syms y1 y2 y3
syms z1 z2 z3
syms w1 w2 w3
syms d d1
syms a1 b1 a2 b2
clc
 %Calculo Algebraico.
eq1 = (x-x1)^2 + (y-y1)^2 + (z-z1)^2 - r^2
eq2 = (x-x2)^2 + (y-y2)^2 + (z-z2)^2 - r^2
eq3 = (x-x3)^2 + (y-y3)^2 + (z-z3)^2 - r^2

eq4 = (eq1 - eq2)
eq5 = (eq1 - eq3)
eq6 = (eq2 - eq3)

eq4_2 = expand(eq4);
eq4_2 = subs(eq4_2, x1^2 + y1^2 + z1^2, w1);
eq4_2 = subs(eq4_2, - x2^2 - y2^2 - z2^2, w2);
eq4_2 = collect(eq4_2,[x,y,z])

eq5_2 = expand(eq5);
eq5_2 = subs(eq5_2, x1^2 + y1^2 + z1^2, w1);
eq5_2 = subs(eq5_2, - x3^2 - y3^2 - z3^2, w3);
eq5_2 = collect(eq5_2,[x,y,z])

eq6_2 = expand(eq6);
eq6_2 = subs(eq6_2, x2^2 + y2^2 + z2^2, w2);
eq6_2 = subs(eq6_2, - x3^2 - y3^2 - z3^2, w3);
eq6_2 = collect(eq6_2,[x,y,z])

S = solve(eq4_2, eq5_2, x, y)
solx = S.x;
soly = S.y;

solx = expand(solx);
soly = expand(soly);

solx_z = collect(solx,z)
soly_z = collect(soly,z)

sol_zd = subexpr([solx_z, soly_z],d)
d=collect(d,[x1, x2, x3])

solx_zda1=subs(sol_zd(1),'d*y1*z2-d*y2*z1-d*y1*z3+d*y3*z1+d*y2*z3-d*y3*z2',a1);
soly_zda1=subs(sol_zd(2),'(d*x1*z2 - d*x2*z1 - d*x1*z3 + d*x3*z1 + d*x2*z3 - d*x3*z2)',a2);
solx_zda1b1=subs(solx_zda1,'(d*w1*y2)/2 + (d*w2*y1)/2 - (d*w1*y3)/2 - (d*w3*y1)/2 -
(d*w2*y3)/2 + (d*w3*y2)/2',b1)
soly_zda1b1=subs(soly_zda1,'(d*w1*x3)/2 - (d*w1*x2)/2 - (d*w2*x1)/2 + (d*w3*x1)/2 +
(d*w2*x3)/2 - (d*w3*x2)/2 ',b2)

eq1_solz = subs(eq1, x, solx_zda1b1);
eq1_solz = subs(eq1_solz, y, soly_zda1b1)

sol_z = solve(eq1_solz,z);
sol_x = subs(solx_zda1b1, z, sol_z);
sol_y = subs(soly_zda1b1, z, sol_z);

sol_z(2)
sol_x(1)
sol_y(1)
```

A continuación se muestra la salida del script:

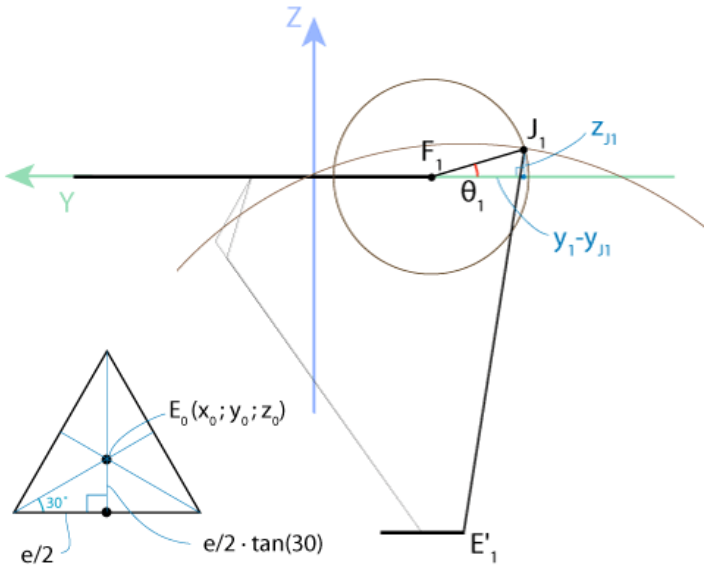
```
eq1 =
 (x - x1)^2 + (y - y1)^2 + (z - z1)^2 - r^2

eq2 =
 (x - x2)^2 + (y - y2)^2 + (z - z2)^2 - r^2

eq3 =
 (x - x3)^2 + (y - y3)^2 + (z - z3)^2 - r^2

eq4 =
 (x - x1)^2 - (x - x2)^2 + (y - y1)^2 - (y - y2)^2 + (z - z1)^2 - (z - z2)^2

eq5 =
 (x - x1)^2 - (x - x3)^2 + (y - y1)^2 - (y - y3)^2 + (z - z1)^2 - (z - z3)^2

eq6 =
 (x - x2)^2 - (x - x3)^2 + (y - y2)^2 - (y - y3)^2 + (z - z2)^2 - (z - z3)^2

eq4_2 =
 (2*x2 - 2*x1)*x + (2*y2 - 2*y1)*y + (2*z2 - 2*z1)*z + w1 + w2

eq5_2 =
 (2*x3 - 2*x1)*x + (2*y3 - 2*y1)*y + (2*z3 - 2*z1)*z + w1 + w3

eq6_2 =
 (2*x3 - 2*x2)*x + (2*y3 - 2*y2)*y + (2*z3 - 2*z2)*z + w2 + w3

sol_zd =

[ z*(d*y1*z2 - d*y2*z1 - d*y1*z3 + d*y3*z1 + d*y2*z3 - d*y3*z2) + (d*w1*y2)/2 + (d*w2*y1)/2 - (d*w1*y3)/2 - (d*w3*y1)/2 - (d*w2*y3)/2 +
(d*w3*y2)/2, (d*w1*x3)/2 - (d*w1*x2)/2 - (d*w2*x1)/2 - z*(d*x1*z2 - d*x2*z1 - d*x1*z3 + d*x3*z1 + d*x2*z3 - d*x3*z2) + (d*w3*x1)/2 + (d*w2*x3)/2 -
(d*w3*x2)/2]

d =
-1/((y3 - y2)*x1 + (y1 - y3)*x2 + (y2 - y1)*x3)

solx_zda1b1 =

b1 + a1*z

soly_zda1b1 =

b2 - a2*z

eq1_solz =

(z - z1)^2 + (b1 - x1 + a1*z)^2 + (y1 - b2 + a2*z)^2 - r^2

ans =
(z1 - a1*b1 + a2*b2 + a1*x1 - a2*y1 - (- a1^2*b2^2 + 2*a1^2*b2*y1 + a1^2*r^2 - a1^2*y1^2 - a1^2*z1^2 - 2*a1*a2*b1*b2 + 2*a1*a2*b1*y1 +
2*a1*a2*b2*x1 - 2*a1*a2*x1*y1 - 2*a1*b1*z1 + 2*a1*x1*z1 - a2^2*b1^2 + 2*a2^2*b1*x1 + a2^2*r^2 - a2^2*x1^2 - a2^2*z1^2 + 2*a2*b2*z1 -
2*a2*y1*z1 - b1^2 + 2*b1*x1 - b2^2 + 2*b2*y1 + r^2 - x1^2 - y1^2)^(1/2))/(a1^2 + a2^2 + 1)


ans =
b1 + (a1*(z1 - a1*b1 + a2*b2 + a1*x1 - a2*y1 + (- a1^2*b2^2 + 2*a1^2*b2*y1 + a1^2*r^2 - a1^2*y1^2 - a1^2*z1^2 - 2*a1*a2*b1*b2 + 2*a1*a2*b1*y1 +
2*a1*a2*b2*x1 - 2*a1*a2*x1*y1 - 2*a1*b1*z1 + 2*a1*x1*z1 - a2^2*b1^2 + 2*a2^2*b1*x1 + a2^2*r^2 - a2^2*x1^2 - a2^2*z1^2 + 2*a2*b2*z1 -
2*a2*y1*z1 - b1^2 + 2*b1*x1 - b2^2 + 2*b2*y1 + r^2 - x1^2 - y1^2)^(1/2)))/(a1^2 + a2^2 + 1)


ans =
b2 - (a2*(z1 - a1*b1 + a2*b2 + a1*x1 - a2*y1 + (- a1^2*b2^2 + 2*a1^2*b2*y1 + a1^2*r^2 - a1^2*y1^2 - a1^2*z1^2 - 2*a1*a2*b1*b2 + 2*a1*a2*b1*y1 +
2*a1*a2*b2*x1 - 2*a1*a2*x1*y1 - 2*a1*b1*z1 + 2*a1*x1*z1 - a2^2*b1^2 + 2*a2^2*b1*x1 + a2^2*r^2 - a2^2*x1^2 - a2^2*z1^2 + 2*a2*b2*z1 -
2*a2*y1*z1 - b1^2 + 2*b1*x1 - b2^2 + 2*b2*y1 + r^2 - x1^2 - y1^2)^(1/2)))/(a1^2 + a2^2 + 1)
```

Las últimas tres respuestas son respectivamente z, x e y, en función a las caracterisitcas constructivas del robot y los angulos Theta1, Theta2 y Theta3.

Cinemática inversa

La cinemática inversa tiene por fin conocer el movimiento que cada actuador debe imprimir en las articulaciones motorizadas para una determinada posición de la punta de nuestro robot; en nuestro caso E0.

Por lo tanto para la resolución de estas ecuaciones se trabaja observando un solo brazo del robot trabajando sobre un plano y teniendo en cuenta todas las consideraciones anteriormente mencionadas.



$$E(x_0, y_0, z_0)$$

$$EE_1 = \frac{e}{2}\tan 30° = \frac{e}{2\sqrt{3}}$$

$$E_1(x_0, y_0 - \frac{e}{2\sqrt{3}}, z_0) \Rightarrow E'_1(0, y_0 - \frac{e}{2\sqrt{3}}, z_0)$$

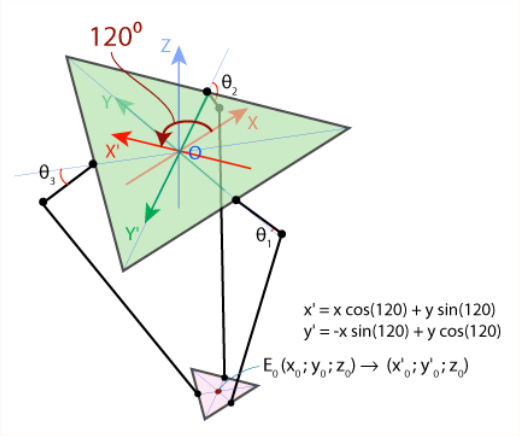
$$E_1E'_1 = x_0 \Rightarrow E'_1J_1 = \sqrt{E_1J_1^{\,2} - E_1E'^{\,2}_1} = \sqrt{r_e^2 - x_0^2}$$

$$F_1(0, -\frac{f}{2\sqrt{3}}, 0)$$

$$\begin{cases} (y_{J1} - y_{F1})^2 + (z_{J1} - z_{F1})^2 = r_f^2 \\ (y_{J1} - y_{F'1})^2 + (z_{J1} - z_{F'1})^2 = r_e^2 - x_0^2 \end{cases} \Rightarrow \begin{cases} (y_{J1} + \frac{f}{2\sqrt{3}})^2 + z_{J1}^{\,2} = r_f^2 \\ (y_{J1} - y_0 + \frac{e}{2\sqrt{3}})^2 + (z_{J1} - z_0)^2 = r_e^2 - x_0^2 \end{cases} \Rightarrow J_1(0, y_{J1}, z_{J1})$$

$$\theta_1 = \arctan\left(\frac{z_{J1}}{y_{F1} - y_{J1}}\right)$$

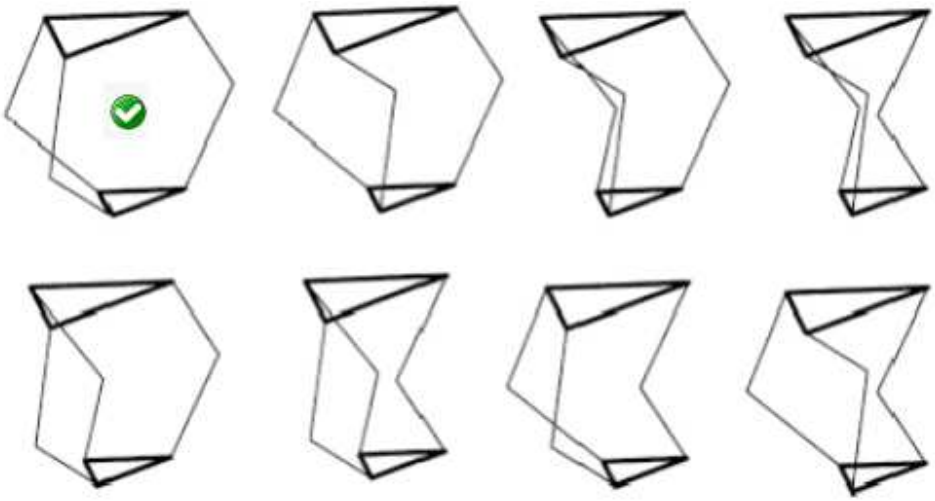
Esta simplicidad algebráica viene dada de optar por un buen sistema de referencia. Dicho sistema de referencia debe ser rotado por una matriz de rotación sobre el eje z rotandolo 120° para obtener así los restantes águnos Theta2 y Theta3; como se ejemplifica en la siguiente figura:



Se propone el siguiente codigo en C para utilizar y obtener la posición de los respectivos ángulos:

```
 /**Inverse Kinematics
  *helper functions, calculates angle theta1 (for YZ-pane)
  */

int delta_calcAngleYZ(float x0, float y0, float z0, float &theta)
{
    float y1 = -0.5 * 0.57735 * f;    /* f/2 * tg 30 */
    y0 -= 0.5 * 0.57735    * e;       /* shift center to edge */
    /* z = a + b*y */
    float a = (x0*x0 + y0*y0 + z0*z0 +rf*rf - re*re - y1*y1)/(2*z0);
    float b = (y1-y0)/z0;
    float d = -(a+b*y1)*(a+b*y1)+rf*(b*b*rf+rf);
    if (d < 0) return -1;                                   /* non-existing point    */
    float yj = (y1 - a*b - sqrt(d))/(b*b + 1);              /* choosing outer point*/
    float zj = a + b*yj;
    theta = 180.0*atan(-zj/(y1 - yj))/pi + ((yj>y1)?180.0:0.0);
    return 0;
}

/* inverse kinematics: (x0, y0, z0) -> (theta1, theta2, theta3)
  * returns:
  *         status: 0=OK
  *                 -1=non-existing position
  */
 int delta_calcInverse(float x0, float y0, float z0, float &theta1, float &theta2, float &theta3)
{
    theta1 = theta2 = theta3 = 0;
    int status = delta_calcAngleYZ(x0, y0, z0, theta1);
    if (status == 0) status = delta_calcAngleYZ(x0*cos120 + y0*sin120, y0*cos120-x0*sin120, z0, theta2);
    if (status == 0) status = delta_calcAngleYZ(x0*cos120 - y0*sin120, y0*cos120+x0*sin120, z0, theta3);
    return status;
}
```

Es preciso aclarar que para cada una de las articulaciones, se pueden obtener dos posibles resultados. Por este motivo se obtendrán 8 (resultados^{articulaciones}=2^3) combinaciones de ternas θ_1, θ_2, θ_3 que resuelven la ecuación de cinemática inversa. Esta situación se puede ver esquematizada en la Ilustración 5. En nuestro caso, sólo nos quedaremos con la solución que satisface: θ_1, θ_2, $\theta_3 \in$ [90°,-90°], la cual se encuentra marcada en la figura.



Simulación utilizando Matlab

A continuación se desarrolla el código utilizado para hacer un diagrama en tres dimensiones del espacio de trabajo del robot. Para ello se iteró múltiples veces sobre un rango de valores para las tres primeras variables articulares del robot.

```
function resultado=cinematica_directa(configuracion,angulo1,angulo2,angulo3)

e = configuracion(1); %115.0;
f = configuracion(2); %457.3;
re = configuracion(3); %232.0;
rf = configuracion(4); %112.0;

sqrt3 = sqrt(3.0);
sin120 = sqrt3/2.0;
cos120 = -0.5;
tan60 = sqrt3;
sin30 = 0.5;
tan30 = 1/sqrt3;

t = (f-e)*tan30/2;
dtr = pi/180.0;

angulo1=angulo1*dtr;
angulo2=angulo2*dtr;
```

```
angulo3=angulo3*dtr;

y1 = -(t + rf*cos(angulo1));
z1 = -rf*sin(angulo1);

y2 = (t + rf*cos(angulo2))*sin30;
x2 = y2*tan60;
z2 = -rf*sin(angulo2);

y3 = (t + rf*cos(angulo3))*sin30;
x3 = -y3*tan60;
z3 = -rf*sin(angulo3);

dnm = (y2-y1)*x3-(y3-y1)*x2;

w1 = y1*y1 + z1*z1;
w2 = x2*x2 + y2*y2 + z2*z2;
w3 = x3*x3 + y3*y3 + z3*z3;

% x = (a1*z + b1)/dnm
a1 = (z2-z1)*(y3-y1)-(z3-z1)*(y2-y1);
b1 = -((w2-w1)*(y3-y1)-(w3-w1)*(y2-y1))/2.0;

% y = (a2*z + b2)/dnm;
a2 = -(z2-z1)*x3+(z3-z1)*x2;
b2 = ((w2-w1)*x3 - (w3-w1)*x2)/2.0;

% a*z^2 + b*z + c = 0
a = a1*a1 + a2*a2 + dnm*dnm;
b = 2*(a1*b1 + a2*(b2-y1*dnm) - z1*dnm*dnm);
c = (b2-y1*dnm)*(b2-y1*dnm) + b1*b1 + dnm*dnm*(z1*z1 - re*re);

% DISCRIMINANTE
d = b^2 - 4*a*c;
if d < 0
   % NO EXISTE EL PUNTO
   resultado=0;
else
   resultado=zeros(3,1);
   resultado(3) = 0.5*(b+sqrt(d))/a;
   resultado(1) = (a1*resultado(3) + b1)/dnm;
   resultado(2) = (a2*resultado(3) + b2)/dnm;
end
```

Función para obtener los puntos del espacio de trabajo y graficarlos en un área tridimensional:

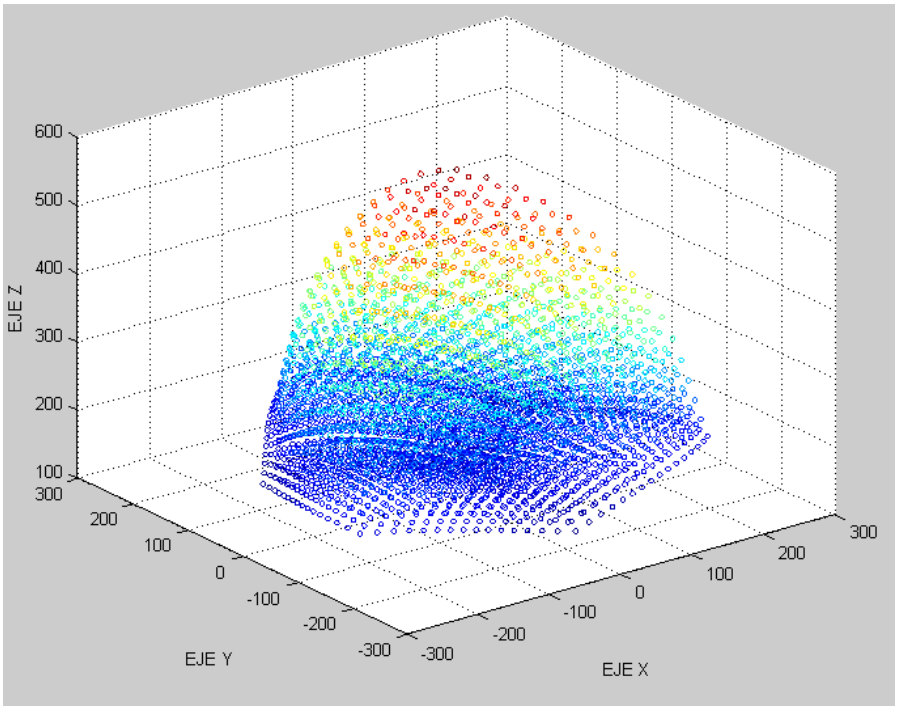
```
function resultado=calcular_area_trabajo(configuracion,angulo_inicial,angulo_final,paso)

resultado=zeros(((angulo_final-angulo_inicial)/paso)^3,3);
indice=1;
for i=angulo_inicial:paso:angulo_final
   for j=angulo_inicial:paso:angulo_final
      for k=angulo_inicial:paso:angulo_final
         resultado_parcial=cinematica_directa(configuracion,i,j,k);
         if resultado_parcial==0
            continue
         end
         resultado(indice,1)=resultado_parcial(1);
         resultado(indice,2)=resultado_parcial(2);
         resultado(indice,3)=resultado_parcial(3);
         indice=indice+1;
      end
   end
end
scatter3(resultado(:,1),resultado(:,2),resultado(:,3),8,resultado(:,3));
```

Una vez obtenidas las funciones necesarias para lograr un gráfico del espacio de trabajo del robot, se obtuvo el mismo, para la siguiente configuración:

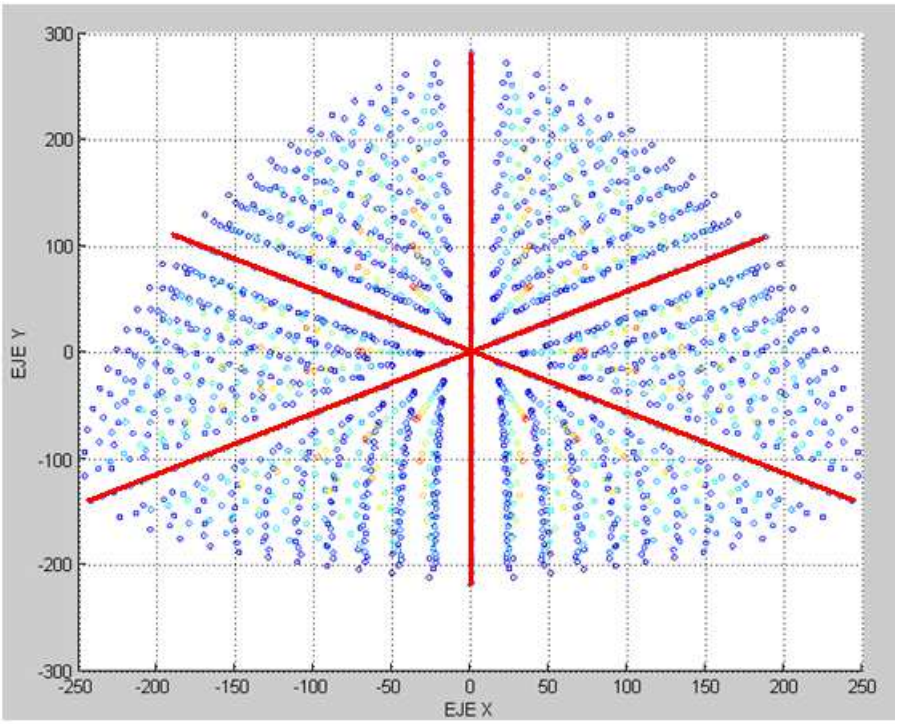
- f=270mm
- e=100mm
- rf(l)=250mm
- re(b)=400mm
- $\theta_1, \theta_2, \theta_3 \in [-60°, 60°]$

Nota: estos valores fueron elegidos a priori tratando de obtener un espacio de trabajo lineal de 450mm aproximadamente (aplicación: pick and place). Se pueden consultar diversas bibliografías que explican metodologías para la obtención de dimensiones óptimas.

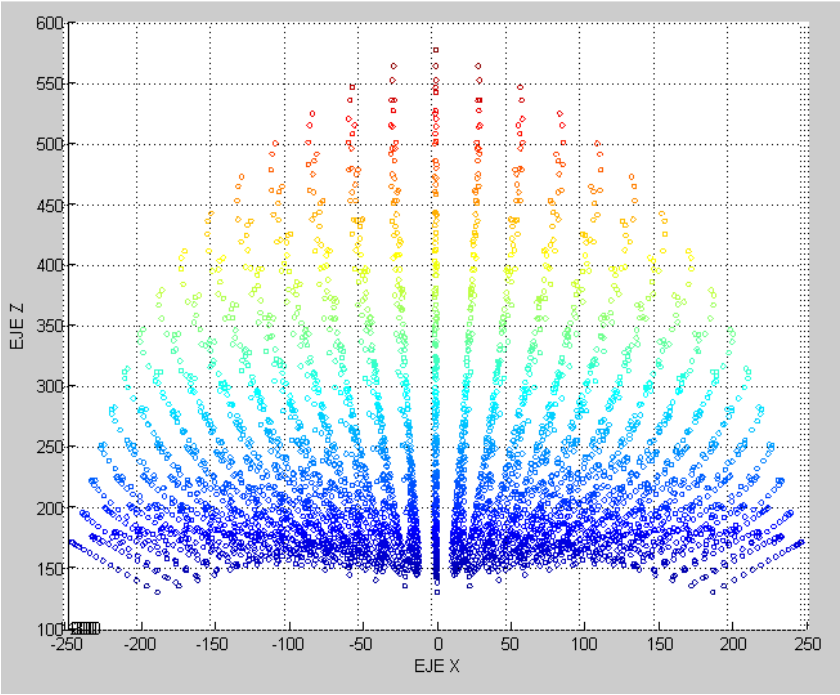
Se muestran a continuación los gráficos obtenidos (unidades de los gráficos: mm):



Espacio de trabajo de la configuración descripta



Vista superior del espacio de trabajo. En rojo, los segmentos de máxima longitud de la vista

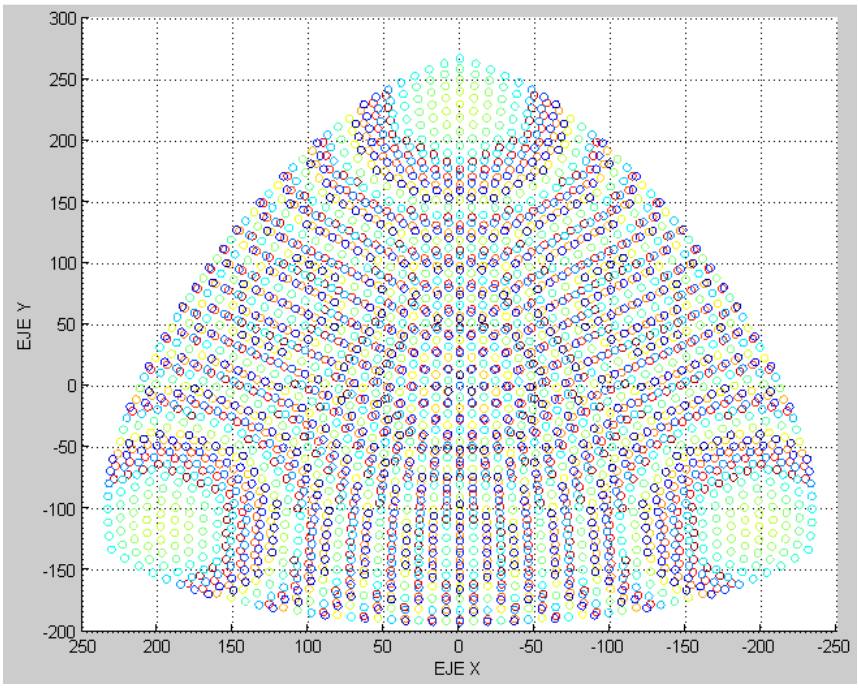


Vista lateral del espacio de trabajo

Para corroborar que en altura Z en la que se moverá el extremo del robot, el mismo puede llegar a todos los puntos de espacio de trabajo requerido, se puede hacer un corte del espacio con un plano perpendicular al eje Z a la altura elegida. Para esto se puede realizar un script como el siguiente:

```
>> resultado=calcular_area_trabajo([100 270 400 250],-60,60,4);
>> rows=(resultado(:,3)>215 & resultado(:,3)<235);
>> scatter3(resultado(rows,1),resultado(rows,2),resultado(rows,3),25,resultado(rows,3));
```

En el mismo vemos que el corte se realiza a la altura Z=225±10mm. A continuación se muestra una vista superior de este corte:



Vista superior de un corte del espacio de trabajo por un plano perpendicular al eje Z (Z=225±10mm)

De esta manera podemos afirmar que con la configuración elegida se puede obtener el espacio de trabajo requerido.

Simulación utilizando un DSP.

Para la realizar una implementación real de un robot es necesario contar con una electrónica asociada capaz de calcular y realizar operaciones matemáticas como por ejemplo multiplicación o evaluación de una función trigonométrica. Por lo tanto para poseer dicha capacidad de cálculo se opta por la utilización de un DSP. Sin embargo, como alternativa de menor calidad para la evaluación de funciones trigonométricas se pueden utilizar tablas de "lookup" como técnica de programación de sistemas embebidos para agilizar dichas operaciones, siempre y cuando se conozcan todas las entradas (rango y precisión) que tendrán las evaluaciones. Por otro lado no existe una forma sencilla de lograr operaciones con números de coma flotante, por lo que la solución de un DSP se adapta mejor a esta necesidad. Para la realización del presente trabajo se utilizará un DSP56800/E de Freescale.

A continuación se puede ver el código para obtener el espacio del robot:

```
#include "cinematica_delta.h"
#include "stdio.h"
```

```c
#include "string.h"

#define PASO            5
#define VALOR_MAXIMO    60
#define VALOR_MINIMO    -60

void main()
{
    FILE* ptrFile = fopen( "espacio_trabajo.csv" , "a" );

    if(ptrFile == (void*)0)
    {
        printf("No se pudo crear el archivo: %s","") ;
        return;
    }

    CargarDimensiones(270,100,250,400);

    int escrituras=0;
    int t1,t2,t3;
    float x,y,z;
    char lineaArchivo[100];

    for(t1=VALOR_MINIMO;t1<VALOR_MAXIMO;t1+=PASO)
        for(t2=VALOR_MINIMO;t2<VALOR_MAXIMO;t2+=PASO)
            for(t3=VALOR_MINIMO;t3<VALOR_MAXIMO;t3+=PASO)
            {
                CinematicaDirecta(t1,t2,t3,&x,&y,&z);
                sprintf(lineaArchivo,"%f,%f,%f\r\n",x,y,z);
                fputs(lineaArchivo,ptrFile) ;
                escrituras++;
            }

    fclose(ptrFile);
}
```

//cinematica_delta.h

```c
void CargarDimensiones(float valor_f,float valor_e,float valor_rf,float valor_re);

int CinematicaDirecta(float theta1, float theta2, float theta3, float* x0, float* y0,
float* z0);
int CinematicaInversa(float x0, float y0, float z0, float* theta1, float* theta2, float*
theta3);
```

//cinematica_delta.c

```c
const float sin120 = sqrt3/2.0;
const float cos120 = -0.5;
const float tan60 = sqrt3;
const float sin30 = 0.5;
const float tan30 = 1/sqrt3;

float e; // = 115.0;
float f; // = 457.3;
float re; // = 232.0;
float rf; // = 112.0;

void CargarDimensiones(float valor_f,float valor_e,float valor_rf,float valor_re)
{
    f=valor_f;
    e=valor_e;
    rf=valor_rf;
    re=valor_re;
}
```

```
// forward kinematics: (theta1, theta2, theta3) -> (x0, y0, z0)
// returned status: 0=OK, -1=non-existing position
int CinematicaDirecta(float theta1, float theta2, float theta3, float* x0, float* y0,
float* z0) {
    float t = (f-e)*tan30/2;
    float dtr = pi/(float)180.0;

    theta1 *= dtr;
    theta2 *= dtr;
    theta3 *= dtr;

    float y1 = -(t + rf*cos(theta1));
    float z1 = -rf*sin(theta1);

    float y2 = (t + rf*cos(theta2))*sin30;
    float x2 = y2*tan60;
    float z2 = -rf*sin(theta2);

    float y3 = (t + rf*cos(theta3))*sin30;
    float x3 = -y3*tan60;
    float z3 = -rf*sin(theta3);

    float dnm = (y2-y1)*x3-(y3-y1)*x2;

    float w1 = y1*y1 + z1*z1;
    float w2 = x2*x2 + y2*y2 + z2*z2;
    float w3 = x3*x3 + y3*y3 + z3*z3;

    // x = (a1*z + b1)/dnm
    float a1 = (z2-z1)*(y3-y1)-(z3-z1)*(y2-y1);
    float b1 = -((w2-w1)*(y3-y1)-(w3-w1)*(y2-y1))/2.0;

    // y = (a2*z + b2)/dnm;
    float a2 = -(z2-z1)*x3+(z3-z1)*x2;
    float b2 = ((w2-w1)*x3 - (w3-w1)*x2)/2.0;

    // a*z^2 + b*z + c = 0
    float a = a1*a1 + a2*a2 + dnm*dnm;
    float b = 2*(a1*b1 + a2*(b2-y1*dnm) - z1*dnm*dnm);
    float c = (b2-y1*dnm)*(b2-y1*dnm) + b1*b1 + dnm*dnm*(z1*z1 - re*re);

    // discriminant
    float d = b*b - (float)4.0*a*c;
    if (d < 0) return -1; // non-existing point

    (*z0) = -(float)0.5*(b+sqrt(d))/a;
    (*x0) = (a1*(*z0) + b1)/dnm;
    (*y0) = (a2*(*z0) + b2)/dnm;
    return 0;
}

 // inverse kinematics
 // helper functions, calculates angle theta1 (for YZ-pane)
int delta_calcAngleYZ(float x0, float y0, float z0, float* theta) {
    float y1 = -0.5 * 0.57735 * f; // f/2 * tg 30
    y0 -= 0.5 * 0.57735    * e;    // shift center to edge
    // z = a + b*y
    float a = (x0*x0 + y0*y0 + z0*z0 +rf*rf - re*re - y1*y1)/(2*z0);
    float b = (y1-y0)/z0;
    // discriminant
    float d = -(a+b*y1)*(a+b*y1)+rf*(b*b*rf+rf);
    if (d < 0) return -1; // non-existing point
    float yj = (y1 - a*b - sqrt(d))/(b*b + 1); // choosing outer point
    float zj = a + b*yj;
    (*theta) = 180.0*atan(-zj/(y1 - yj))/pi + ((yj>y1)?180.0:0.0);
    return 0;
}
```

```
 // inverse kinematics: (x0, y0, z0) -> (theta1, theta2, theta3)
 // returned status: 0=OK, -1=non-existing position
int CinematicaInversa(float x0, float y0, float z0, float* theta1, float* theta2, float*
theta3) {
    theta1 = theta2 = theta3 = 0;
    int status = delta_calcAngleYZ(x0, y0, z0, theta1);
    if (status == 0) status = delta_calcAngleYZ(x0*cos120 + y0*sin120, y0*cos120-x0*sin120,
z0, theta2);  // rotate coords to +120 deg
    if (status == 0) status = delta_calcAngleYZ(x0*cos120 - y0*sin120, y0*cos120+x0*sin120,
z0, theta3);  // rotate coords to -120 deg
    return status;
}
```

Conclusiones

- Se pudo corroborar el funcionamiento del robot delta.
- Se comprendió exaustivamente la matemática asociada a dicho robot y como es el comportamiento general del robot.
- Se pudo obtener una ecuación para cada coordenada de la cinematica directa que luego se pudo llevar a un código escrito en C.
- Se pudo obtener y comprender la ecuación de la cinemática inversa.
- Conclusiones indirectas:
    - Se obtuvo una muy buena experiencia en el trabajo de operaciones algebráicas.
    - Se obtuvo una mayor experiencia en el manejo de Matlab.
    - Se aprendió a utilizar el IDE CodeWarrior.

Mejoras

- Analizar el comportamiento en situaciones críticas y detectar por código los puntos de colisión.

Referencias y bibliografía

- Modeling and control of a Delta-3 robot.
  André Olsson
- robotkinematics-http://forums.trossenrobotics.com/tutorials/introduction-129/delta-robot-kinematics-3276/
- Design optimization, Impedance Control and Characterization of a Modified Delta Robot.
  Mehmet Alper ERGIN, Aykut Cihan SATICI, Volkan PATOGLU
- Delta robot: inverse, direct, and intermediate Jacobians.
  M. López, E. Castillo, G. García y A. Bashir.
- An Improved Approach to the Kinematics of Clavel's DELTA Robot.

P.J. Zsombor-Murray Center for Intelligent Machines, McGill University, Montreal, Canada-November 27, 2009

- The mathematical model and direct kinematics solution analysis of Delta parallel robot.
  Jingjun Zhang -Lihong Shi Ruizhen Gao Chaoyang Lian
- TOWARDS A FULLY-PARALLEL 6 DOF ROBOT FOR HIGH-SPEED APPLICATIONS.
  F. Pierrot, A. Fournier and P. Dauchez
- A New Approach to the Design of a DELTA Robot with a Desired Workspace.
  XIN-JUN LIU-JINSONGWANG-KUN-KU OH and JONGWON KIM