```
clear
close all
clc

% Declaración de variables simbólicas
syms theta(t) z(t) y(t) t

% Configuración del robot (0 = rotacional, 1 = prismática)
RP = [0 1 1];

% Coordenadas articulares
Q = [theta, z, y];

% Velocidades generalizadas (derivadas de las coordenadas articulares)
Qp = diff(Q, t);

% Número de grados de libertad (GDL)
GDL = length(RP);

% Matrices de transformación homogénea
I3 = eye(3);
zeroRow = [0 0 0 1];

% Transformaciones homogéneas
T = sym(zeros(4,4,GDL));
A = sym(zeros(4,4,GDL));

% Articulación 1: Rotación en Z (θ)
Rz = [cos(theta) -sin(theta) 0; sin(theta) cos(theta) 0; 0 0 1];
P1 = [0; 0; 0]; % No traslación
A(:,:,1) = [Rz P1; zeroRow];

% Articulación 2: Movimiento en Z (z)
P2 = [0; 0; z];
A(:,:,2) = [I3 P2; zeroRow];

% Articulación 3: Movimiento en Y (y)
P3 = [0; y; 0];
A(:,:,3) = [I3 P3; zeroRow];

% Matriz de transformación global
for i = 1:GDL
    if i == 1
        T(:,:,i) = A(:,:,i);
    else
        T(:,:,i) = T(:,:,i-1) * A(:,:,i);
    end

    % Mostrar matrices
    disp(['Matriz de Transformación Global T', num2str(i)]);
```

```
    pretty(T(:,:,i));
end
```

```
Matriz de Transformación Global T1
/ cos(theta(t)), -sin(theta(t)), 0, 0 \
|                                     |
| sin(theta(t)),  cos(theta(t)), 0, 0 |
|                                     |
|       0,              0,       1, 0 |
|                                     |
\       0,              0,       0, 1 /
Matriz de Transformación Global T2
/ cos(theta(t)), -sin(theta(t)), 0,   0  \
|                                        |
| sin(theta(t)),  cos(theta(t)), 0,   0  |
|                                        |
|       0,              0,       1, z(t) |
|                                        |
\       0,              0,       0,   1  /
Matriz de Transformación Global T3
/ cos(theta(t)), -sin(theta(t)), 0, -sin(theta(t)) y(t) \
|                                                       |
| sin(theta(t)),  cos(theta(t)), 0,  cos(theta(t)) y(t) |
|                                                       |
|       0,              0,       1,         z(t)        |
|                                                       |
\       0,              0,       0,          1          /
```

```
% Cálculo del Jacobiano
Jv_a = sym(zeros(3, GDL));
Jw_a = sym(zeros(3, GDL));

% Posición del extremo del robot
P_extremo = T(1:3, 4, GDL);

for k = 1:GDL
    if RP(k) == 0 % Si es rotacional
        % Jacobiano lineal: producto cruz entre z_k y (P_extremo - O_k)
        z_k = T(1:3,3,k); % Vector z de la articulación
        Jv_a(:,k) = cross(z_k, P_extremo);
        Jw_a(:,k) = z_k; % Rotación en torno a z_k
    else % Si es prismática
        Jv_a(:,k) = T(1:3,3,k); % Z_k da la dirección de movimiento
        Jw_a(:,k) = [0; 0; 0]; % No genera velocidad angular
    end
end

% Simplificación del Jacobiano
Jv_a = simplify(Jv_a);
Jw_a = simplify(Jw_a);

% Velocidades lineales y angulares
V = simplify(Jv_a * Qp.');
W = simplify(Jw_a * Qp.');
```

```
% Mostrar resultados
disp('Jacobiano Lineal:');
```

```
Jacobiano Lineal:
```

```
pretty(Jv_a);
```

```
/ -cos(theta(t)) y(t), 0, 0 \
|                           |
| -sin(theta(t)) y(t), 0, 0 |
|                           |
\          0,          1, 1 /
```

```
disp('Jacobiano Angular:');
```

```
Jacobiano Angular:
```

```
pretty(Jw_a);
```

```
/ 0, 0, 0 \
|         |
| 0, 0, 0 |
|         |
\ 1, 0, 0 /
```

```
disp('Velocidad Lineal:');
```

```
Velocidad Lineal:
```

```
pretty(V);
```

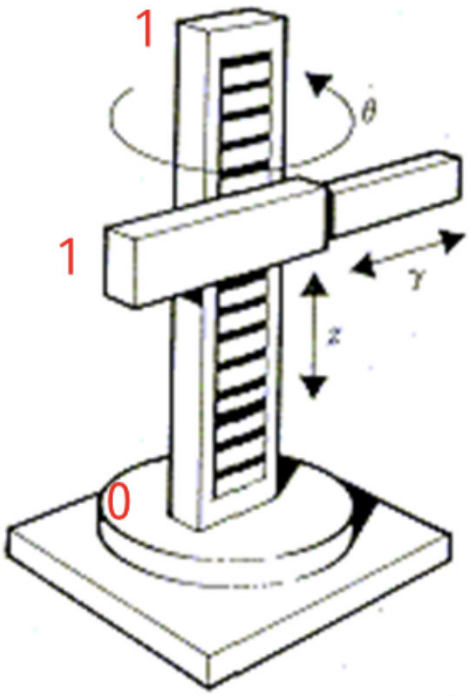
```
/                      d          \
| -cos(theta(t)) y(t) -- theta(t) |
|                      dt         |
|                                 |
|                      d          |
| -sin(theta(t)) y(t) -- theta(t) |
|                      dt         |
|                                 |
|         d         d             |
|        -- y(t) + -- z(t)        |
\        dt        dt            /
```

```
disp('Velocidad Angular:');
```

```
Velocidad Angular:
```

```
pretty(W);
```

```
/      0      \
|             |
|      0      |
|             |
|  d          |
| -- theta(t) |
\ dt          /
```


1
θ
1
r
z
0