In [ ]:
```
%matplotlib widget

import sympy as sp
import sympy.physics.mechanics as me
import sympy.plotting as splt
from typing import List
from sympy import sin, cos, pi, sqrt, acos, simplify, atan
import math
me.init_vprinting()


def homogeneous(rotation: sp.Matrix = sp.eye(3), translation: sp.Matrix = sp
    return rotation.row_join(translation).col_join(sp.Matrix([[0, 0, 0, 1]])

def dh(rotation, twist, displacement, offset):
    rotation_mat = sp.Matrix([
        [cos(rotation), -sin(rotation)*cos(twist),  sin(rotation)*sin(twist)
        [sin(rotation), cos(rotation)*cos(twist),   -cos(rotation)*sin(twist
        [0,             sin(twist),                 cos(twist)],
    ])
    translation = sp.Matrix([
        [offset*cos(rotation)],
        [offset*sin(rotation)],
        [displacement],
    ])
    return rotation_mat, translation

def rotation(homogeneous: sp.Matrix):
    return homogeneous[:3, :3]

def translation(homogeneous: sp.Matrix):
    return homogeneous[:3, 3:]

def chained_transform(transforms: List[sp.Matrix]):
    transforms_chained = [homogeneous()]
    for transform in transforms:
        transforms_chained.append(transforms_chained[-1] * transform)
    return transforms_chained

def z_vecs(transforms: List[sp.Matrix]):
    transforms_chained = chained_transform(transforms)
    z_unit_vecs = []
    for transform in transforms_chained:
        z_unit_vecs.append(rotation(transform) * sp.Matrix([0, 0, 1]))
    return z_unit_vecs

def jacobian(transforms: List[sp.Matrix], joint_types: List[sp.Matrix], base
    transforms_chained = chained_transform(transforms)
    z_unit_vecs = z_vecs(transforms)

    assert len(transforms_chained) == len(z_unit_vecs)

    jacobian = sp.zeros(6, len(transforms))
    for i, (transform, joint_type) in enumerate(zip(transforms, joint_types)
```

```
        if joint_type == 'revolute':
            jacobian[:3, i] = z_unit_vecs[i].cross(translation(transforms_ch
            jacobian[3:, i] = z_unit_vecs[i]
        elif joint_type == 'prismatic':
            jacobian[:3, i] = z_unit_vecs[i]
            jacobian[3:, i] = sp.Matrix([[0], [0], [0]])

        # angular velocity

    return jacobian

def skew(v: sp.Matrix):
    return sp.Matrix([
        [0, -v[2], v[1]],
        [v[2], 0, -v[0]],
        [-v[1], v[0], 0],
    ])
```

In [ ]:
```
t = sp.symbols('t')
g = sp.symbols('g')
# joint variables
# theta_1, d_1 = sp.symbols('\\theta_1, d_1')
theta_1, d_2 = me.dynamicsymbols('\\theta_1, d_2')
q = sp.Matrix([theta_1, d_2])
q_dot = q.diff(t)
# physical properties
l_c1, m_1, m_2, I_1, I_2 = sp.symbols('l_c1, m_1, m_2, I_1, I_2')
r_c = [sp.Matrix([0, 0, 0, 1]), sp.Matrix([0, 0, l_c1, 1]), sp.Matrix([0, 0,
m = [None, m_1, m_2]
I = [None, I_1, I_2]

joint_1 = homogeneous(*dh(pi/2 + theta_1, pi/2, 0, 0))
joint_2 = homogeneous(*dh(0, 0, d_2, 0))
all_joints = [joint_1, joint_2]
joint_types = ['revolute', 'prismatic']
```

In [ ]:
```
display(joint_1)
display(joint_2)
display(joint_1*sp.Matrix([0, 0, 0, 1]))
display(joint_1*joint_2)
```

$$\begin{bmatrix} -\sin(\theta_1) & 0 & \cos(\theta_1) & 0 \\ \cos(\theta_1) & 0 & \sin(\theta_1) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} -\sin(\theta_1) & 0 & \cos(\theta_1) & d_2\cos(\theta_1) \\ \cos(\theta_1) & 0 & \sin(\theta_1) & d_2\sin(\theta_1) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In [ ]:
```
# Compute Jacobian
J = jacobian(all_joints, joint_types)

w = [sp.Matrix([0, 0, 0])]*(len(all_joints)+1) # joint linear velocities
v = [sp.Matrix([0, 0, 0])]*(len(all_joints)+1) # joint angular velocities
v_c = [sp.Matrix([0, 0, 0])]*(len(all_joints)+1) # joint CoM linear velociti
T = [sp.Matrix([0])]*(len(all_joints)+1) # joint kinetic energy
V = [sp.Matrix([0])]*(len(all_joints)+1) # joint potential energy

chained_transforms = chained_transform(all_joints)
# chained_translations = chained_translation(all_joints)
# z = z_vecs(all_joints) # joint origins
z = sp.Matrix([0, 0, 1]) # base z vector

for i, joint, joint_type in zip(range(1, len(all_joints) + 1), all_joints, j
    # Compute angular velocity
    theta_dot = q_dot[i-1] if joint_type == 'revolute' else 0
    w[i] = rotation(joint).T * (w[i-1] + z*theta_dot)

    # Compute linear velocity
    d_dot = q_dot[i-1] if joint_type == 'prismatic' else 0
    r_i = (joint*sp.Matrix([0, 0, 0, 1]))[:3, :]
    v[i] = rotation(joint).T * (v[i-1] + z*d_dot) + w[i].cross(r_i)

    # Compute CoM linear velocity
    v_c[i] = v[i] + w[i].cross(r_c[i][:3, :])

    # Compute kinetic energy
    T[i] = 0.5*m[i]*v_c[i].T*v_c[i] + 0.5*w[i].T*I[i]*w[i]

    # Compute potential energy
    p_ci = (chained_transforms[i]*r_c[i])[:3, :]
    V[i] = -m[i]*sp.Matrix([0, -g, 0]).T*p_ci

display(J)
display(w[1])
display(w[2])
display(v[1])
display(v[2])
display(v_c[1])
display(v_c[2])
display(T[1])
display(T[2])
```

```
display(V[1])
display(V[2])
```

$$\begin{bmatrix} -d_2 \sin(\theta_1) & \cos(\theta_1) \\ d_2 \cos(\theta_1) & \sin(\theta_1) \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ \dot{\theta}_1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ \dot{\theta}_1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} d_2\dot{\theta}_1 \\ 0 \\ \dot{d}_2 \end{bmatrix}$$

$$\begin{bmatrix} l_{c1}\dot{\theta}_1 \\ 0 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} d_2\dot{\theta}_1 \\ 0 \\ \dot{d}_2 \end{bmatrix}$$

$$\begin{bmatrix} 0.5I_1\dot{\theta}_1^2 + 0.5l_{c1}^2 m_1\dot{\theta}_1^2 \end{bmatrix}$$

$$\begin{bmatrix} 0.5I_2\dot{\theta}_1^2 + 0.5m_2d_2^2\dot{\theta}_1^2 + 0.5m_2\dot{d}_2^2 \end{bmatrix}$$

$$\begin{bmatrix} gl_{c1}m_1 \sin(\theta_1) \end{bmatrix}$$

$$\begin{bmatrix} gm_2d_2 \sin(\theta_1) \end{bmatrix}$$

In [ ]:

```
# Construct robot dynamics
T = T[1]+T[2]
V = V[1]+V[2]
L = T - V

f_x, f_y, f_z, g_x, g_y, g_z = sp.symbols('f_x, f_y, f_z, g_x, g_y, g_z')
F_ext = sp.Matrix([f_x, f_y, f_z, g_x, g_y, g_z])
```

```
# to = L.diff(q_dot).diff(t) - L.diff(q)

temp = L.diff(q_dot).diff(t) - L.diff(q)
temp = sp.Matrix([temp[0][0][0][0], temp[1][0][0][0]])
temp - J.T*F_ext
```

Out[ ]:

$$\left[\begin{matrix} 1.0 I_1 \ddot{\theta}_1 + 1.0 I_2 \ddot{\theta}_1 + f_x d_2 \sin(\theta_1) - f_y d_2 \cos(\theta_1) + g l_{c1} m_1 \cos(\theta_1) + g m_2 d_2 \cos(\theta_1) \\ -f_x \cos(\theta_1) - f_y \sin(\theta_1) + g m_2 \sin(\theta_1) - 1.0 m_2 d \end{matrix}\right.$$