Kinematic Model

Jacobian Calculations 4 Bar JupyterLab

In [2]:
```
import sympy
from sympy import sin,cos
from math import pi
import numpy
import matplotlib.pyplot as plt
import scipy.optimize as so
```

In [3]:
```
q1,q2,q3,q4 = sympy.symbols('\\theta_1,\\theta_2,\\theta_3,\\theta_4')
```

In [4]:
```
q1
```

Out[4]: θ_1

In [5]:
```
l0,l1,l2,l3 = sympy.symbols('l_0,l_1,l_2,l_3')
l1
```

Out[5]: l_1

In [6]:
```
def Ry(theta):
    return sympy.Matrix([[cos(theta),0,sin(theta)],
                         [0,1,0],
                         [-sin(theta),0,cos(theta)]])
```

In [7]:
```
Ra = Ry(q1)
Rb = Ry(q2)
Rc = Ry(q3)
Rd = Ry(q4)
Ra
```

Out[7]:
$$\begin{bmatrix} \cos(\theta_1) & 0 & \sin(\theta_1) \\ 0 & 1 & 0 \\ -\sin(\theta_1) & 0 & \cos(\theta_1) \end{bmatrix}$$

In [8]:
```
x = sympy.Matrix([1,0,0])
```

In [9]:
```
v0_in_n=l0*x
#v1_in_n=R
v0_in_n
```

Out[9]:
$$\begin{bmatrix} l_0 \\ 0 \\ 0 \end{bmatrix}$$

In [10]:
```
v1_in_a = l1*x
v1_in_n = Ra*v1_in_a
v1_in_n
```

Out[10]:
$$\begin{bmatrix} l_1 \cos(\theta_1) \\ 0 \\ -l_1 \sin(\theta_1) \end{bmatrix}$$

In [11]:
```
v2_in_b = l2*x
v2_in_a = Rb*v2_in_b
v2_in_n = Ra*v2_in_a
v2_in_n
```

Out[11]:
$$\begin{bmatrix} -l_2 \sin(\theta_1)\sin(\theta_2) + l_2 \cos(\theta_1)\cos(\theta_2) \\ 0 \\ -l_2 \sin(\theta_1)\cos(\theta_2) - l_2 \sin(\theta_2)\cos(\theta_1) \end{bmatrix}$$

In [12]:
```
v3_in_c = l3*x
v3_in_b = Rc*v3_in_c
v3_in_a = Rb*v3_in_b
v3_in_n = Ra*v3_in_a
v3_in_n
```

Out[12]:
$$\begin{bmatrix} (-l_3 \sin(\theta_2)\sin(\theta_3) + l_3 \cos(\theta_2)\cos(\theta_3))\cos(\theta_1) + (-l_3 \sin(\theta_2)\cos(\theta_3) - l_3 \sin(\theta_3)\cos(\theta_2)) \\ 0 \\ -(-l_3 \sin(\theta_2)\sin(\theta_3) + l_3 \cos(\theta_2)\cos(\theta_3))\sin(\theta_1) + (-l_3 \sin(\theta_2)\cos(\theta_3) - l_3 \sin(\theta_3)\cos(\theta_2)) \end{bmatrix}$$

In [13]:
```
#v1_in_n.subs({q1:pi*30/180,l1:1})
```

In [14]:
```
p_end = v0_in_n + v1_in_n + v2_in_n + v3_in_n
```

In [15]:
```
# LINK GUESS / DESIGN
design = {}
design[l0] = 44
design[l1] = 33.3
design[l2] = 54
design[l3] = 13

design
```

Out[15]:
```
{l_0: 44, l_1: 33.3, l_2: 54, l_3: 13}
```

In [16]:
```
zero_vec = p_end
zero_vec.simplify()
zero_vec[0]
```

Out[16]:
$$l_0 + l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3)$$

In [17]:
```
zero = []
zero.append((zero_vec.T*sympy.Matrix([1,0,0]))[0])
zero.append((zero_vec.T*sympy.Matrix([0,1,0]))[0])
```

In [18]:
```
zero = sympy.Matrix(zero)
zero.simplify()
```

```
zero[0]
```

Out[18]: $l_0 + l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3)$

In [19]:
```
zero_design = zero.subs(design)
zero_design = zero_design
zero_design
```

Out[19]: $\begin{bmatrix} 33.3\cos(\theta_1) + 54\cos(\theta_1 + \theta_2) + 13\cos(\theta_1 + \theta_2 + \theta_3) + 44 \\ 0 \end{bmatrix}$

In [20]:
```
def objective_function(qn):
    q1n,q2n,q3n,q4n=qn
    subs = {}
    subs[q1]=q1n
    subs[q2]=q2n
    subs[q3]=q3n
    subs[q4]=q4n
    zero_n = zero_design.subs(subs)
    sos = ((zero_n.T*zero_n)[0])**.5
    return float(sos)
```

In [21]:
```
# THETA guesses
guess = numpy.array([-76,-126,-60,-81])*pi/180
guess
```

Out[21]:
```
array([-1.32645023, -2.19911486, -1.04719755, -1.41371669])
```

In [22]:
```
objective_function(guess)
```

Out[22]:
```
0.17882066438156152
```

In [23]:
```
origin = sympy.Matrix([0,0,0])
p0 = v0_in_n
p1 = v0_in_n + v1_in_n
p2 = v0_in_n + v1_in_n + v2_in_n
p3 = v0_in_n + v1_in_n + v2_in_n + v3_in_n
```

In [24]:
```
points = sympy.Matrix([p3.T,p2.T,p1.T,p0.T,origin.T]).T
points
```

Out[24]: $\begin{bmatrix} l_0 + l_1\cos(\theta_1) - l_2\sin(\theta_1)\sin(\theta_2) + l_2\cos(\theta_1)\cos(\theta_2) + (-l_3\sin(\theta_2)\sin(\theta_3) + l_3\cos(\theta_2)\cos(\\ 0 \\ -l_1\sin(\theta_1) - l_2\sin(\theta_1)\cos(\theta_2) - l_2\sin(\theta_2)\cos(\theta_1) - (-l_3\sin(\theta_2)\sin(\theta_3) + l_3\cos(\theta_2)\cos(\theta \end{bmatrix}$

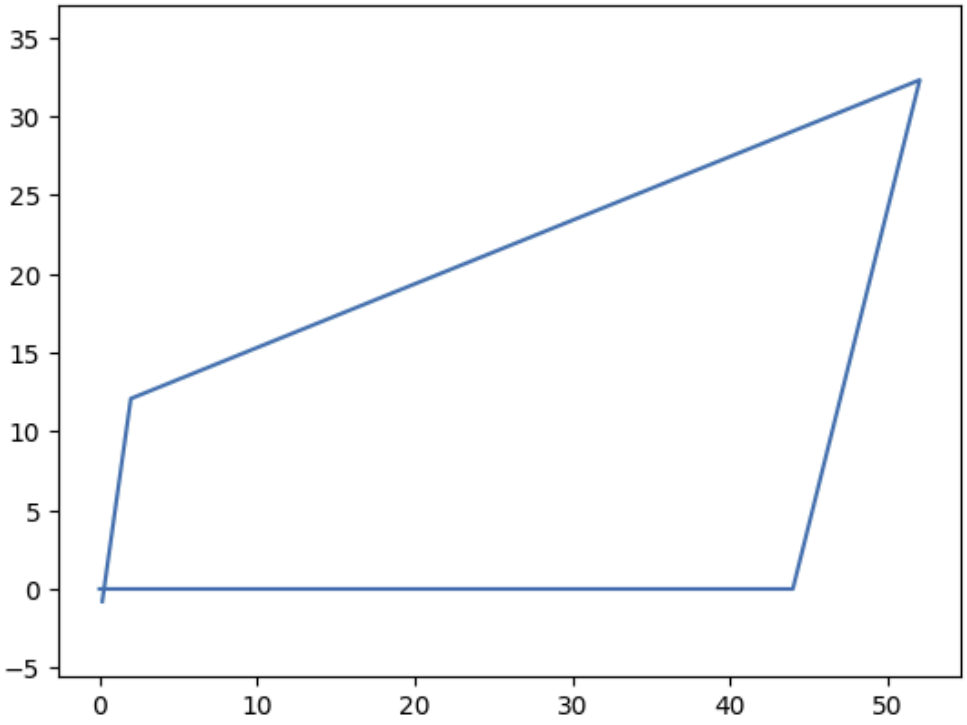
In [25]:
```
points_design = points.subs(design)
```

In [26]:
```
def plot_fourbar(thetas):
    state_variables = q1,q2,q3,q4
    state = dict(zip(state_variables,thetas))
    points_state = points_design.subs(state)
    points_state_numpy = numpy.array(points_state,dtype=numpy.float64)
```

```
    print(points_state_numpy)
    plt.plot(points_state_numpy[0, :], points_state_numpy[2, :])
    plt.axis('equal')

plot_fourbar(guess) # FINAL MODEL
```

```
[[ 0.17882066  1.98807098 52.05599912 44.          0.        ]
 [ 0.          0.          0.          0.          0.        ]
 [-0.79139325 12.08209164 32.31084768  0.          0.        ]]
```



In [27]:
```
zero = sympy.Matrix(zero)
zero.simplify()
zero
```

Out[27]:
$$\begin{bmatrix} l_0 + l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2) + l_3 \cos(\theta_1 + \theta_2 + \theta_3) \\ 0 \end{bmatrix}$$

In [28]:
```
independent = sympy.Matrix([q1])
dependent = sympy.Matrix([q1,q2,q3,q4])
zero_design = zero.subs(design)
```

In [29]:
```
A = zero_design.jacobian(independent)
A
```

Out[29]:
$$\begin{bmatrix} -33.3 \sin(\theta_1) - 54 \sin(\theta_1 + \theta_2) - 13 \sin(\theta_1 + \theta_2 + \theta_3) \\ 0 \end{bmatrix}$$

In [30]:
```
B = zero_design.jacobian(dependent)
B
```

Out[30]:
$$\begin{bmatrix} -33.3\sin(\theta_1) - 54\sin(\theta_1+\theta_2) - 13\sin(\theta_1+\theta_2+\theta_3) & -54\sin(\theta_1+\theta_2) - 13\sin(\theta_1+\theta_2+\theta_3) \\ 0 & 0 \end{bmatrix}$$

In [31]:
```
result1 = so.minimize(objective_function,guess,options={'xrtol':1e-4})
state_variables = q1,q2,q3,q4
state = dict(zip(state_variables,result1.x))
An = numpy.array(A.subs(state),dtype=float)
An
```

Out[31]:
```
array([[-0.53848029],
       [ 0.        ]])
```

In [32]:
```
Bn = numpy.array(B.subs(state),dtype=float)
Bn
```

Out[32]:
```
array([[ -0.53848029, -32.84842135, -12.86117031,   0.        ],
       [  0.        ,   0.        ,   0.        ,   0.        ]])
```

In [33]:
```
print(Bn.shape)
```

```
(2, 4)
```

In [34]:
```
import numpy as np
Cn, *_ = np.linalg.lstsq(Bn, -An, rcond=None)
Cn
```

Out[34]:
```
array([[-0.00023295],
       [-0.01421063],
       [-0.0055639 ],
       [ 0.        ]])
```

In [35]:
```
p_out =v3_in_n
p_out
```

Out[35]:
$$\begin{bmatrix} (-l_3\sin(\theta_2)\sin(\theta_3) + l_3\cos(\theta_2)\cos(\theta_3))\cos(\theta_1) + (-l_3\sin(\theta_2)\cos(\theta_3) - l_3\sin(\theta_3)\cos(\theta_2)) \\ 0 \\ -(-l_3\sin(\theta_2)\sin(\theta_3) + l_3\cos(\theta_2)\cos(\theta_3))\sin(\theta_1) + (-l_3\sin(\theta_2)\cos(\theta_3) - l_3\sin(\theta_3)\cos(\theta_2)) \end{bmatrix}$$

In [36]:
```
D = p_out.jacobian(independent)
Dn = numpy.array(D.subs(design).subs(state),dtype=float)
E = p_out.jacobian(dependent)
En = numpy.array(E.subs(design).subs(state),dtype=float)
Jo = Dn+(En@Cn)
Jo
```

Out[36]:
```
array([[-12.60385074],
       [  0.        ],
       [  1.85689791]])
```

In [37]:
```
y_dot = Jo@numpy.array([[1,]]).T
y_dot
```

```
Out[37]: array([[-12.60385074],
                [  0.        ],
                [  1.85689791]])

In [38]: p_out_n = numpy.array(p_out.subs(design).subs(state),dtype=float)
         p_out_n

Out[38]: array([[ -1.89480825],
                [  0.        ],
                [-12.86117031]])

In [ ]:
```

Another code for jacobian:

```python
import sympy as sp
from sympy import cos, sin, sqrt, atan2, simplify, diff

# Define symbols
theta1, theta2, theta3, theta4 = sp.symbols('theta1 theta2 theta3 theta4', real=True)
l1, l2, l3, l4, l_E = sp.symbols('l1 l2 l3 l4 l_E', positive=True)

print("="*70)
print("4-BAR LINKAGE - SYMBOLIC POSITION OF POINT E")
print("="*70)

# Closure equations (for reference)
print("\nClosure equations:")
eq1 = l2*cos(theta2) + l3*cos(theta3) - l1*cos(theta1) - l4*cos(theta4)
eq2 = l2*sin(theta2) + l3*sin(theta3) - l1*sin(theta1) - l4*sin(theta4)
print("Eq1:", eq1, "= 0")
print("Eq2:", eq2, "= 0")

# Position of point E
# Path: Origin -> Link 1 base -> Link 3 base -> Point E on Link 3
print("\n" + "="*70)
print("POSITION OF POINT E (SYMBOLIC)")
print("="*70)

print("\nPath to Point E:")
print("  Origin (0,0) -> Link 1 endpoint -> Along Link 3 by distance l_E")

# Link 1 endpoint (base of link 3)
x_link3_base = l1*cos(theta1)
y_link3_base = l1*sin(theta1)

print(f"\nLink 3 base position:")
print(f"  x_base = l1*cos(θ1)")
print(f"  y_base = l1*sin(θ1)")

# Point E is at distance l_E from link 3 base, along link 3 direction (θ3)
x_E = x_link3_base + l_E*cos(theta3)
y_E = y_link3_base + l_E*sin(theta3)
```

```
print(f"\nPoint E position:")
print(f"  x_E = l1*cos(θ1) + l_E*cos(θ3)")
print(f"  y_E = l1*sin(θ1) + l_E*sin(θ3)")

print("\nSimplified:")
print(f"  x_E = {x_E}")
print(f"  y_E = {y_E}")

# Since θ1 is fixed at 135°, we can express E purely in terms of θ3
print("\n" + "="*70)
print("WITH θ1 = 135° (FIXED)")
print("="*70)

theta1_val = sp.pi * 3 / 4  # 135° in radians

x_E_fixed = x_E.subs(theta1, theta1_val)
y_E_fixed = y_E.subs(theta1, theta1_val)

print(f"\nPosition with θ1 = 135°:")
print(f"  x_E = {x_E_fixed}")
print(f"  y_E = {y_E_fixed}")

# Now compute Jacobian: ∂E/∂θ2
# Since E depends on θ3, and θ3 depends on θ2, we use chain rule
print("\n" + "="*70)
print("JACOBIAN ∂E/∂θ2 (SYMBOLIC)")
print("="*70)

print("\nUsing chain rule:")
print("  ∂x_E/∂θ2 = ∂x_E/∂θ3 · ∂θ3/∂θ2")
print("  ∂y_E/∂θ2 = ∂y_E/∂θ3 · ∂θ3/∂θ2")

# Partial derivatives of E with respect to θ3
dx_E_dtheta3 = diff(x_E_fixed, theta3)
dy_E_dtheta3 = diff(y_E_fixed, theta3)

print(f"\n∂x_E/∂θ3 = {dx_E_dtheta3}")
print(f"∂y_E/∂θ3 = {dy_E_dtheta3}")
```

```
# For ∂θ3/∂θ2, we need to differentiate the constraint equations
print("\n" + "="*70)
print("FINDING ∂θ3/∂θ2 FROM CONSTRAINT EQUATIONS")
print("="*70)

print("\nDifferentiating closure equations with respect to θ2:")

# Differentiate eq1 and eq2 with respect to theta2
deq1_dtheta2 = diff(eq1, theta2) + diff(eq1, theta3)*diff(theta3, theta2) + diff(eq1, theta4)*diff(theta4, theta2)
deq2_dtheta2 = diff(eq2, theta2) + diff(eq2, theta3)*diff(theta3, theta2) + diff(eq2, theta4)*diff(theta4, theta2)

print("These give us a linear system in ∂θ3/∂θ2 and ∂θ4/∂θ2")

# Let's denote dtheta3_dtheta2 and dtheta4_dtheta2 as unknowns
dtheta3_dtheta2, dtheta4_dtheta2 = sp.symbols('dtheta3_dtheta2 dtheta4_dtheta2', real=True)

# Coefficients from differentiation
coeff_eq1_theta3 = -l3*sin(theta3)
coeff_eq1_theta4 = l4*sin(theta4)
coeff_eq1_theta2 = -l2*sin(theta2)

coeff_eq2_theta3 = l3*cos(theta3)
coeff_eq2_theta4 = -l4*cos(theta4)
coeff_eq2_theta2 = l2*cos(theta2)

print(f"\nFrom Eq1: {coeff_eq1_theta3}·∂θ3/∂θ2 + {coeff_eq1_theta4}·∂θ4/∂θ2 = {-coeff_eq1_theta2}")
print(f"From Eq2: {coeff_eq2_theta3}·∂θ3/∂θ2 + {coeff_eq2_theta4}·∂θ4/∂θ2 = {-coeff_eq2_theta2}")

# Solve the linear system
sol = sp.solve([
    coeff_eq1_theta3*dtheta3_dtheta2 + coeff_eq1_theta4*dtheta4_dtheta2 + coeff_eq1_theta2,
    coeff_eq2_theta3*dtheta3_dtheta2 + coeff_eq2_theta4*dtheta4_dtheta2 + coeff_eq2_theta2
], [dtheta3_dtheta2, dtheta4_dtheta2])

print("\n" + "="*70)
print("SOLUTION FOR ∂θ3/∂θ2")
print("="*70)
print(f"\n∂θ3/∂θ2 = {simplify(sol[dtheta3_dtheta2])}")
```

```
# Final Jacobian
dx_E_dtheta2 = simplify(dx_E_dtheta3 * sol[dtheta3_dtheta2])
dy_E_dtheta2 = simplify(dy_E_dtheta3 * sol[dtheta3_dtheta2])

print("\n" + "="*70)
print("FINAL JACOBIAN EQUATIONS")
print("="*70)
print(f"\n∂x_E/∂θ2 = {dx_E_dtheta2}")
print(f"\n∂y_E/∂θ2 = {dy_E_dtheta2}")

print("\n" + "="*70)
print("SUMMARY")
print("="*70)
print("\nPosition of Point E:")
print(f"  x_E = {x_E_fixed}")
print(f"  y_E = {y_E_fixed}")
print("\nJacobian components:")
print(f"  ∂x_E/∂θ2 = {dx_E_dtheta2}")
print(f"  ∂y_E/∂θ2 = {dy_E_dtheta2}")
print("="*70)
```

✓ 1.0s

```
======================================================================
4-BAR LINKAGE - SYMBOLIC POSITION OF POINT E
======================================================================

Closure equations:
Eq1: -l1*cos(theta1) + l2*cos(theta2) + l3*cos(theta3) - l4*cos(theta4) = 0
Eq2: -l1*sin(theta1) + l2*sin(theta2) + l3*sin(theta3) - l4*sin(theta4) = 0

======================================================================
POSITION OF POINT E (SYMBOLIC)
======================================================================

Path to Point E:
  Origin (0,0) -> Link 1 endpoint -> Along Link 3 by distance l_E

Link 3 base position:
  x_base = l1*cos(θ1)
  y_base = l1*sin(θ1)

Point E position:
  x_E = l1*cos(θ1) + l_E*cos(θ3)
  y_E = l1*sin(θ1) + l_E*sin(θ3)

Simplified:
  x_E = l1*cos(theta1) + l_E*cos(theta3)
  y_E = l1*sin(theta1) + l_E*sin(theta3)

======================================================================
WITH θ1 = 135° (FIXED)
======================================================================

Position with θ1 = 135°:
  x_E = -sqrt(2)*l1/2 + l_E*cos(theta3)
  y_E = sqrt(2)*l1/2 + l_E*sin(theta3)

======================================================================
JACOBIAN ∂E/∂θ2 (SYMBOLIC)
======================================================================

Using chain rule:
  ∂x_E/∂θ2 = ∂x_E/∂θ3 · ∂θ3/∂θ2
  ∂y_E/∂θ2 = ∂y_E/∂θ3 · ∂θ3/∂θ2
```

```
∂x_E/∂θ3 = -l_E*sin(theta3)
∂y_E/∂θ3 = l_E*cos(theta3)

======================================================================
FINDING ∂θ3/∂θ2 FROM CONSTRAINT EQUATIONS
======================================================================

Differentiating closure equations with respect to θ2:
These give us a linear system in ∂θ3/∂θ2 and ∂θ4/∂θ2

From Eq1: -l3*sin(theta3)·∂θ3/∂θ2 + l4*sin(theta4)·∂θ4/∂θ2 = l2*sin(theta2)
From Eq2: l3*cos(theta3)·∂θ3/∂θ2 + -l4*cos(theta4)·∂θ4/∂θ2 = -l2*cos(theta2)

======================================================================
SOLUTION FOR ∂θ3/∂θ2
======================================================================

∂θ3/∂θ2 = -l2*sin(theta2 - theta4)/(l3*sin(theta3 - theta4))

======================================================================
FINAL JACOBIAN EQUATIONS
======================================================================

∂x_E/∂θ2 = l2*l_E*sin(theta3)*sin(theta2 - theta4)/(l3*sin(theta3 - theta4))

∂y_E/∂θ2 = -l2*l_E*sin(theta2 - theta4)*cos(theta3)/(l3*sin(theta3 - theta4))
```

```
======================================================================
SUMMARY
======================================================================

Position of Point E:
  x_E = -sqrt(2)*l1/2 + l_E*cos(theta3)
  y_E = sqrt(2)*l1/2 + l_E*sin(theta3)

Jacobian components:
  ∂x_E/∂θ2 = l2*l_E*sin(theta3)*sin(theta2 - theta4)/(l3*sin(theta3 - theta4))
  ∂y_E/∂θ2 = -l2*l_E*sin(theta2 - theta4)*cos(theta3)/(l3*sin(theta3 - theta4))
======================================================================
```