6ELEN018W - Tutorial 3 2026 Solutions

[1]:
```python
from sympy import *
from roboticstoolbox import *
from spatialmath.base import *
import numpy as np
```

Exercise 1

[2]:
```python
q1, q2, q3, q4, a1, a2, a3, a4 = symbols('q1, q2, q3, q4, a1, a2, a3, a4')

R = trot2(q1)@transl2(a1, 0)@trot2(q2)@transl2(a2, 0)@trot2(q3)@transl2(a3,␣
↪0)@trot2(q4)@transl2(a4, 0)
simplify(R)
```

[2]:

$$\begin{bmatrix} \cos(q_1+q_2+q_3+q_4) & -\sin(q_1+q_2+q_3+q_4) & a_1\cos(q_1)+a_2\cos(q_1+q_2)+a_3\cos(q_1+q_2+q_3)+a_4\cos(q_1 \\ \sin(q_1+q_2+q_3+q_4) & \cos(q_1+q_2+q_3+q_4) & a_1\sin(q_1)+a_2\sin(q_1+q_2)+a_3\sin(q_1+q_2+q_3)+a_4\sin(q_1 \\ 0 & 0 & 1 \end{bmatrix}$$

Exercise 2

[3]:
```python
import math

def ex2(q1, q2, a1, a2):
    r1 = np.array([[cos(q1), -sin(q1), 0],
                   [sin(q1), cos(q1),  0],
                    [0,       0,       1]])
    t1 = np.array([[1, 0, a1],
                   [0, 1, 0],
                   [0, 0, 1]])
    r2 = np.array([[cos(q2), -sin(q2), 0],
                   [sin(q2), cos(q2), 0],
                   [0, 0, 1]])
    t2 = np.array([[1, 0, a2],
                   [0, 1, 0],
                   [0, 0, 1]])

    return r1@t1@r2@t2
```

```
# calling the function
print(ex2(math.pi, math.pi/2, 2, 3))

# toolbox equivalent
trot2(math.pi)@transl2(2,0)@trot2(math.pi/2)@transl2(3,0)
```

```
[[-1.83697019872103e-16 1.00000000000000 -2.00000000000000]
 [-1.00000000000000 -1.83697019872103e-16 -3.00000000000000]
 [0 0 1]]
```

[3]:
```
array([[-1.8369702e-16,  1.0000000e+00, -2.0000000e+00],
       [-1.0000000e+00, -1.8369702e-16, -3.0000000e+00],
       [ 0.0000000e+00,  0.0000000e+00,  1.0000000e+00]])
```

[4]:
```
def end_effector():
    tr = ex2(math.pi, math.pi/2, 2, 3)
    print(tr[0,2], tr[1,2])

# calling the function
end_effector()
```

```
-2.00000000000000 -3.00000000000000
```

Exercise 3

[5]:
```
T = trot2(math.pi/2)@transl2(2,0)@trot2(math.pi)@transl2(3,0)@trot2(math.
 ↪pi)@transl2(4,0)

T[0,2], T[1,2]
```

[5]:
```
(7.960204194457794e-16, 3.0)
```

Exercise 4

[6]:
```
def ex4(q1, q2, a1, a2):
    # From equation (8) in the lecture slides
    J = [[-a1*sin(q1)-a2*sin(q1+q2), -a2*sin(q1+q2)],
         [a1*cos(q1)+a2*cos(q1+q2),   a2*cos(q1+q2)]]

    return J

# calling the function with desired angular velocities
q1dot = 4
q2dot = 5
ex4(math.pi/2, math.pi/4, 2, 3)@np.array([q1dot, q2dot])
```

[6]:
```
array([-27.0918830920368, -19.0918830920368], dtype=object)
```

Exercise 5

[7]:
```
from math import *

def ex5(q1, q2, a1, a2, desired_v_list):
    # From equation (8) in the lecture slides
    J = [[-a1*sin(q1)-a2*sin(q1+q2), -a2*sin(q1+q2)],
         [a1*cos(q1)+a2*cos(q1+q2),   a2*cos(q1+q2)]]

    ang_vel = np.linalg.inv(J)@desired_v_list

    return ang_vel

# testing the function
[q1dot, q2dot] = ex5(math.pi/2, math.pi/4, 2, 3, [-27.0918830920368, -19.
↪0918830920368])
print([q1dot, q2dot])
```

```
[4.000000000000001, 5.00000000000001]
```

Exercise 6

[8]:
```
theta = Symbol('theta')
a = Symbol('a')

e1 = trot2(theta)@transl2(a, 0)
e2 = transl2(a, 0)@trot2(theta)
print(e1)
print(e2)
```

```
[[cos(theta) -sin(theta) a*cos(theta)]
 [sin(theta) cos(theta) a*sin(theta)]
 [0 0 1]]
[[cos(theta) -sin(theta) a]
 [sin(theta) cos(theta) 0]
 [0 0 1]]
```

Exercise 7

[9]:
```
import numpy as np
from sympy import *

t = Symbol('t')
a1, a2 = symbols('a1 a2')

q1 = Function('q1')
q2 = Function('q2')
```

```python
#print(t*q1(t))

tr1 = [[cos(q1(t)), -sin(q1(t)), 0],
       [sin(q1(t)), cos(q1(t)), 0],
       [0,         0,      1]]

tr2 = [[1, 0, a1],
       [0, 1, 0],
       [0, 0, 1]]

tr3 = [[cos(q2(t)), -sin(q2(t)), 0],
       [sin(q2(t)), cos(q2(t)), 0],
       [0,         0,      1]]

tr4 = [[1, 0, a2],
       [0, 1, 0],
       [0, 0, 1]]


tr1 = np.array(tr1)
tr2 = np.array(tr2)
tr3 = np.array(tr3)
tr4 = np.array(tr4)

# do the calculation for the end-effector
E = tr1@tr2@tr3@tr4

E = simplify(E)

print(f'x = {E[0, 2]}')
print(f'y = {E[1, 2]}')

# velocities for the end-effector
v_x = diff(E[0,2], t)
v_y = diff(E[1,2], t)

print(f'\n\nVelocities for the end-effector are:\nv_x = {v_x}')
print(f'v_y = {v_y}')
```

```
x = a1*cos(q1(t)) + a2*cos(q1(t) + q2(t))
y = a1*sin(q1(t)) + a2*sin(q1(t) + q2(t))


Velocities for the end-effector are:
v_x = -a1*sin(q1(t))*Derivative(q1(t), t) - a2*(Derivative(q1(t), t) +
Derivative(q2(t), t))*sin(q1(t) + q2(t))
v_y = a1*cos(q1(t))*Derivative(q1(t), t) + a2*(Derivative(q1(t), t) +
```

```
Derivative(q2(t), t))*cos(q1(t) + q2(t))
```

Alternative Solution 2: Assuming the steps up to the calculation of E are the same:

[10]:
```
Matrix([E[0, 2], E[1, 2]]).jacobian([q1(t), q2(t)]) # built-in Jacobian for
↪SymPy Matrix
```

[10]:
$$\begin{bmatrix}-a_1 \sin\left(q_1(t)\right) - a_2 \sin\left(q_1(t) + q_2(t)\right) & -a_2 \sin\left(q_1(t) + q_2(t)\right) \\ a_1 \cos\left(q_1(t)\right) + a_2 \cos\left(q_1(t) + q_2(t)\right) & a_2 \cos\left(q_1(t) + q_2(t)\right)\end{bmatrix}$$

Alternative Solution 3: Assuming the steps up to the calculation of E are the same:

[11]:
```
# or based on the definition of the Jacobian derivatives
Jrow1 = [diff(E[0, 2], q1(t)), diff(E[0, 2], q2(t))]
Jrow2 = [diff(E[1, 2], q1(t)), diff(E[1, 2], q2(t))]
[Jrow1, Jrow2]
```

[11]:
```
[[-a1*sin(q1(t)) - a2*sin(q1(t) + q2(t)), -a2*sin(q1(t) + q2(t))],
 [a1*cos(q1(t)) + a2*cos(q1(t) + q2(t)), a2*cos(q1(t) + q2(t))]]
```