5ELEN018W - Tutorial 2 2026 Solutions

[1]:
```
import math
import numpy as np
from scipy import linalg, optimize
import matplotlib.pyplot as plt
from spatialmath import *
from spatialmath.base import *
from spatialmath.base import sym
from spatialgeometry import *
from roboticstoolbox import *
```

Exercise 1

[13]:
```
def dof(N, J, m, f_list):
    d = m*(N-1-J) + sum(f_list)
    return d

# testing the function
dof(5, 4, 3, [1, 1, 1, 1])
```

[13]: 4

Exercise 2 - Grubler's Formula

$\text{dof} = m \cdot (N - 1 - J) + \sum_{i=1}^{J} f_i = 3 * (6 - 1 - 7) + 1 * 7 = 1$

Exercise 3 - Python F-Strings

[4]:
```
name = input('Enter your name: ')
print(f'Hello {name.upper()}!')
```

```
Enter your name:  John
```

```
Hello JOHN!
```

Exercise 4 - More on Grubler's Formula

[12]:
```
def grubler():
    joint_types = ['revolute', 'prismatic', 'helical', 'cylindrical',␣
 ↪'universal', 'spherical']

    # dof for the joint types above - this is not used here but for reference␣
 ↪it is included
    dof_per_joint_type = [1, 1, 1, 2, 2, 3]

    # this will store user inputs
    number_of_joints_per_type = []
    for joint in joint_types:
        number = input(f'How many {joint} joints: ')
        number_of_joints_per_type.append(int(number))

    N = input('How many links (including the ground): ')
    type = input('planar or spatial: ')  # this should check for valid answers␣
 ↪as well
    m = 3  # default value
    if type.lower() == 'spatial':
        m = 6
    else:
        m = 3

    # calculate total no of joints
    J = sum(number_of_joints_per_type)
    f = 0
    fi = []
    for i in range(J):
        d = int(input(f'How many dof for joint {i}: '))
        fi.append(d)
        f += d

    # Report the results
    print(f'm = {m} ({type})')
    print(f'J = {J}')
    print(f'N = {N} (including the ground)')
    for i in range(1, J+1):
        print(f'f_{i} = {fi[i-1]}')

    print(f'dof = m*(N-1-J)', end='')  # do not insert a newline at the end
    for i in range(1, J+1):
        print(f' + f_{i}', end='')

    print(f' = {m}*({N}-1-{J}) + ', end='')
```

```python
    for i in range(J-1):
        print(f'{fi[i]} + ', end='')
    print(fi[J-1], end='')
    print(f' = {sum(fi)}')



# test the function
grubler()
```

```
How many revolute joints:  4
How many prismatic joints:  0
How many helical joints:  0
How many cylindrical joints:  0
How many universal joints:  0
How many spherical joints:  0
How many links (including the ground):  5
planar or spatial:  planar
How many dof for joint 0:  1
How many dof for joint 1:  1
How many dof for joint 2:  1
How many dof for joint 3:  1
```

```
m = 3 (planar)
J = 4
N = 5 (including the ground)
f_1 = 1
f_2 = 1
f_3 = 1
f_4 = 1
dof = m*(N-1-J) + f_1 + f_2 + f_3 + f_4 = 3*(5-1-4) + 1 + 1 + 1 + 1 = 4
```

0.1 Exercise 5 - A Robot Navigating a Maze

[1]:
```python
import random

maze = {}
# initialise maze as empty
for i in range(1,6):
    for j in range(1,6):
        maze[(i,j)] = ''

# place the obstacles
maze[(2,2)] = 'O'
maze[(3,3)] = 'O'
maze[(4,4)] = 'O'

print(maze)
```

```python
def move(coordinates_list, action):
    x = coordinates_list[0]
    y = coordinates_list[1]

    if action == 'E':
        if y < 5 and maze[(x, y+1)] != 'O':
            return (x, y+1)
    elif action == 'S':
        if x < 5 and maze[(x+1, y)] != 'O':
            return (x+1, y)
    elif action == 'W':
        if y > 1 and maze[(x, y-1)] != 'O':
            return (x, y-1)
    elif action == 'N':
        if x > 1 and maze[(x-1, y)] != 'O':
            return (x-1, y)

    # if nothing of the above the robot cannot move
    return (x, y)
```

```
{(1, 1): '', (1, 2): '', (1, 3): '', (1, 4): '', (1, 5): '', (2, 1): '', (2, 2):
'O', (2, 3): '', (2, 4): '', (2, 5): '', (3, 1): '', (3, 2): '', (3, 3): 'O',
(3, 4): '', (3, 5): '', (4, 1): '', (4, 2): '', (4, 3): '', (4, 4): 'O', (4, 5):
'', (5, 1): '', (5, 2): '', (5, 3): '', (5, 4): '', (5, 5): ''}
```

[2]:
```python
# maps 1 to 'N', 2 to 'E', 3 to 'S', 4 to 'W'
def num2Action(n):
    if n == 1:
        return 'N'
    elif n == 2:
        return 'E'
    elif n == 3:
        return 'S'
    elif n == 4:
        return 'W'

    return 'No op'  # no operation result


# returns the number of steps
def simulation():
    start_position = (4,2)
    goal_position = (2,4)

    current_pos = start_position
    #print(current_pos)
```

```python
    # number of steps
    steps = 0

    # keep moving until the robot reaches the goal
    while current_pos != goal_position:
        # choose a random action
        rand_int = random.randint(1, 4)
        # map random int to an action
        action = num2Action(rand_int)

        # change the current position
        current_pos = move(current_pos, action)
        steps += 1
        #print(current_pos)

    #print(f'number of steps: {steps}')
    return steps

average_steps = 0
for s in range(10000):
    steps = simulation()
    average_steps += steps

print(f'Average number of steps to reach goal: {average_steps/10000}')
```

```
Average number of steps to reach goal: 132.8307
```

[ ]:

```python

```