ME 639

INTRODUCTION TO ROBOTICS

Mini - Project

Topic – 2R Manipulator

Submitted by:

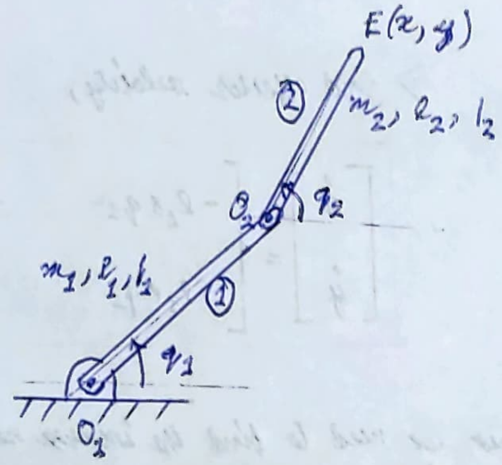
Rhitosparsha Baishya
23310039
PhD (Mechanical Engineering)

CONTENTS

Quick links to the code on GitHub:

1. Task 1
2. Task 2
3. Task 3
4. Task 4

2R MANIPULATOR



Here, m_1, m_2 = Masses of links 1 and 2 respectively

l_1, l_2 = Lengths of links 1 and 2 respectively

I_1, I_2 = Moments of inertia of links 1 and 2 respectively

E = End effector

(x, y) : Coordinates of end effector

O_1 : Origin

q_1, q_2 = Joint angles

We assume that ~~the~~ a motor is connected to each joint O_1 & O_2. Also, we assume that there's a way to control torques applied at the joint τ_1 & τ_2 as well as the joint angles q_1 & q_2.

Now, $x = l_1 \cos q_1 + l_2 \cos q_2$

$y = l_1 \sin q_1 + l_2 \sin q_2$

$$\Rightarrow \boxed{\begin{aligned} x &= l_1 c q_1 + l_2 c q_2 \\ y &= l_1 s q_1 + l_2 s q_2 \end{aligned}} \qquad \text{———— (1)}$$

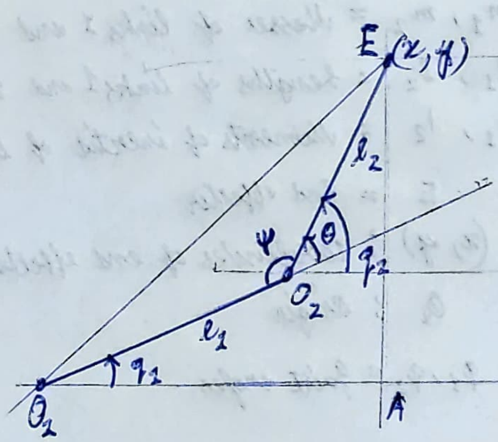
Differentiating (1) w.r.t. time,

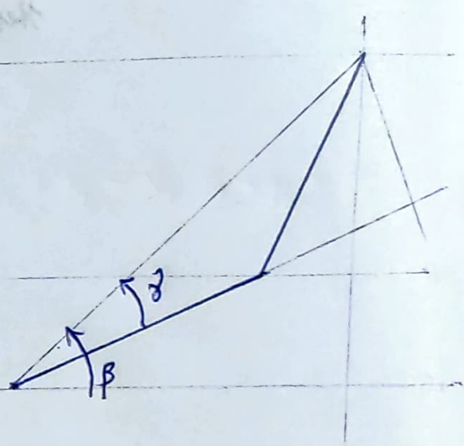
$$\dot{x} = -l_1 s q_1 \dot{q}_1 - l_2 s q_2 \dot{q}_2$$
$$\dot{y} = l_1 c q_1 \dot{q}_1 - l_2 c q_2 \dot{q}_2$$

$\Rightarrow$ End effector velocity,

$$\boxed{\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} -l_1 s q_1 & -l_2 s q_2 \\ l_1 c q_1 & l_2 c q_2 \end{bmatrix} \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \end{bmatrix}} \quad \text{——— (2)}$$

Now, we need to find the inverse relation; i.e. given (x, y), we need to find q_1 & q_2





In $\Delta O_1 A E$, using

$$O_1A^2 + AE^2 = O_1E^2$$
$$\Rightarrow x^2 + y^2 = O_1E^2$$

Using cosine rule in $\Delta O_1 O_2 E$,

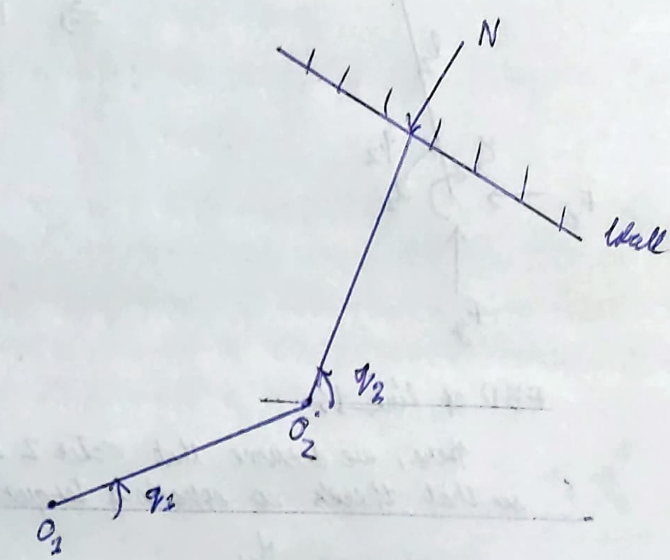
$$O_1E^2 = l_1^2 + l_2^2 - 2 l_1 l_2 \cos\Psi$$
$$\Rightarrow O_1E^2 = l_1^2 + l_2^2 + 2 l_1 l_2 \cos\theta$$

$$\therefore \quad x^2 + y^2 = l_1^2 + l_2^2 + 2 l_1 l_2 \cos\theta$$

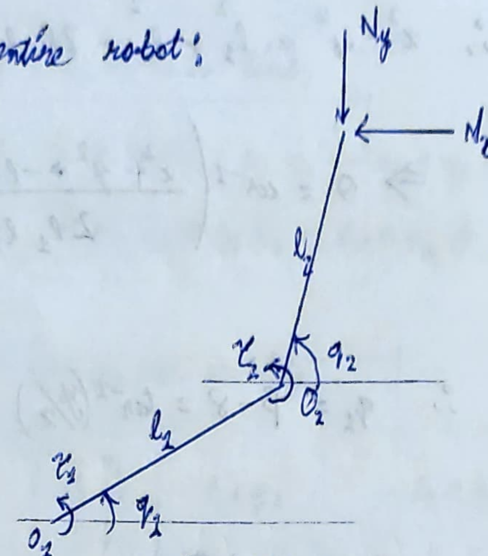
$$\Rightarrow \boxed{\theta = \cos^{-1}\left(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2 l_1 l_2}\right)} \quad \text{———— (3)}$$

$$\therefore \boxed{\begin{aligned} q_1 &= \beta - \gamma = \tan^{-1}(y/x) - \tan^{-1}\left(\frac{l_2 \sin\theta}{l_1 + l_2 \cos\theta}\right) \\ q_2 &= q_1 + \theta \end{aligned}} \quad \text{——(3)}$$

Next, we need to find torques τ_1 and τ_2 so that the robot can apply a normal force N on a wall



FBD of entire robot:



Static equilibrium

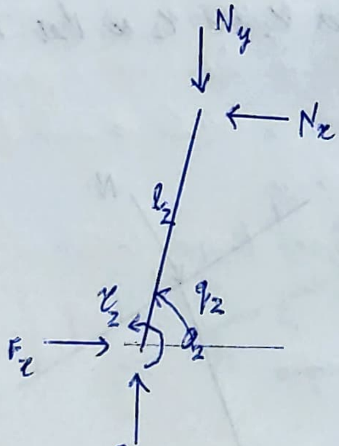
Force applied by manipulator:

$F_x = -N_x$

$F_y = -N_y$

Gravity is neglected

FBD of link 2:

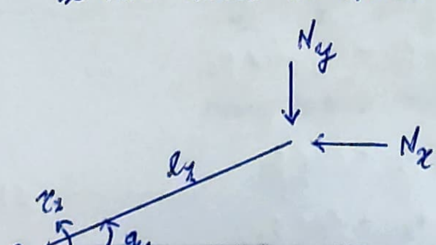


$$\Sigma M_{O_2} = 0$$

$$\Rightarrow N_y l_2 c q_2 - N_x l_2 s q_2 = \tau_2$$

FBD of link 1:

Here, we assume that motor 2 and encoder 2 are on ground, so that there's no opposing torque on link 1.



$$\Sigma M_{O_1} = 0$$

$$\Rightarrow N_y l_1 c q_1 - N_x l_1 s q_1 = \tau_1$$

$$\therefore \boxed{\begin{aligned} N_y\, l_1\, cq_1 - N_x\, l_1\, sq_1 &= \tau_1 \\ N_y\, l_2\, cq_2 - N_x\, l_2\, sq_2 &= \tau_2 \end{aligned}} \quad \text{——— (4)}$$

$$\Rightarrow \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} -l_1\, sq_1 & l_1\, cq_1 \\ -l_2\, sq_2 & l_2\, cq_2 \end{bmatrix} \begin{bmatrix} N_x \\ N_y \end{bmatrix}$$

Now, we introduce dynamics and derive the equations of motion

Lagrangian, $L = K - V$

where K = Kinetic energy

V = Potential energy

We know;

$$\boxed{\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Q_i'} \quad \text{——— (5)}$$

where Q_i' are generalised forces derived using ~~the~~ Principle of Virtual Work

$$K = \underbrace{\tfrac{1}{2}\left(\tfrac{1}{3} m_1 l_1^2\right)\dot{q}_1^2}_{\text{Pure rotation of link 1}} + \underbrace{\tfrac{1}{2}\left(\tfrac{1}{12} m_2 l_2^2\right)\dot{q}_2^2 + \tfrac{1}{2} m_2 v_{c_2}^2}_{\text{Pure rotation of link 2}}$$

$$v_{c_2}^2 = (l_1 \dot{q}_1)^2 + (l_2/2\, \dot{q}_2)^2 + 2\, l_1 \dot{q}_1\, l_2/2\, \dot{q}_2 \cos(q_2 - q_1)$$

$$V = m_1\, g\, l_1/2\, sq_1 + m_2\, g\left(l_1\, sq_1 + l_2/2\, sq_2\right)$$

After solving the above, we get,

$$\boxed{\begin{aligned}\tau_1 = {} & \tfrac{1}{3} m_1 l_1^2 \ddot{q}_1 + m_2 l_1^2 \ddot{q}_1 + m_2 \frac{l_1 l_2}{2} \ddot{q}_2 \cos(q_2 - q_1) \\ & - m_2 \frac{l_1 l_2}{2} \dot{q}_2 (\dot{q}_2 - \dot{q}_1) \sin(q_2 - q_1) + m_1 g \, l_1/2 \, c q_1 + m_2 g l_1 c q_1\end{aligned}}$$

$$\boxed{\begin{aligned}\tau_2 = {} & \tfrac{1}{3} m_2 l_2^2 \ddot{q}_2 + m_2 \, l_2^2/4 \, \ddot{q}_2 + m_2 \frac{l_1 l_2}{2} \ddot{q}_1 \cos(q_2 - q_1) \\ & - m_2 \frac{l_1 l_2}{2} \dot{q}_1 (\dot{q}_2 - \dot{q}_1) \sin(q_2 - q_1) + m_2 g \, l_2/2 \, s q_2\end{aligned}}$$

——————— (6)

TASK 1

Source Code: (Link to GitHub)

```
import numpy as np
import math
from matplotlib import pyplot as plt
from matplotlib.animation import FuncAnimation

l1 = 2.0 # length of link 1 in m
l2 = 2.0 # length of link 2 in m

# here, angles q1 and q2 are both measured from the x-axis

# inverse kinematics
def inv_kin (x, y):
    theta = np.arccos((x**2 + y**2 - l1**2 - l2**2) / (2 * l1 * l2))
    q1 = np.arctan2(y, x) - np.arctan2((l2 * np.sin(theta)), (l1 + l2 *
np.cos(theta)))
    q2 = q1 + theta
    return q1, q2

# forward kinematics
def fwd_kin (q1, q2):
    x = l1 * np.cos(q1) + l2 * np.cos(q2)
    y = l1 * np.sin(q1) + l2 * np.sin(q2)
    return x, y

# create planned trajectory
t_max = 5.0   # total simulation time (s)
num_steps = 50
start_point = (-3.5, 1)
end_point = (-1, 3.5)
trajectory = [(1 - i) * np.array(start_point) + i * np.array(end_point) for i
in np.linspace(0, 1, num_steps)]
joint_angles = [inv_kin(x, y) for x, y in trajectory]

# preparing the plot
fig, ax = plt.subplots(figsize=(8, 8))  # Equal aspect ratio
ax.set_xlim(-4.0, 4.0)
ax.set_ylim(-4.0, 4.0)
ax.scatter(x=0, y=0, c='r')

plt.xlabel('x (m)') #label
plt.ylabel('y (m)')
plt.title('2R Manipulator following a given trajectory')
```

```
plt.grid()

link1, = plt.plot([], [], 'b-')
link2, = plt.plot([], [], 'b-')
point1, = plt.plot([], [], 'r.')
point2, = plt.plot([], [], 'r.')

end_effector_path, = plt.plot([], [], 'g--', label="End-Effector Path")

# animation function
def animate(i):
    q1, q2 = joint_angles[i]
    x, y = fwd_kin(q1, q2)

    link1.set_data([0, l1 * np.cos(q1)], [0, l1 * np.sin(q1)])
    point1.set_data(l1 * np.cos(q1), l1 * np.sin(q1))
    link2.set_data([l1 * np.cos(q1), x], [l1 * np.sin(q1), y])
    point2.set_data(x, y)

    end_effector_path.set_data([p[0] for p in trajectory[:i+1]], [p[1] for p
in trajectory[:i+1]])

ani = FuncAnimation(fig, animate, frames=num_steps, interval=(t_max /
num_steps) * 1000)

plt.show()
```

Output:

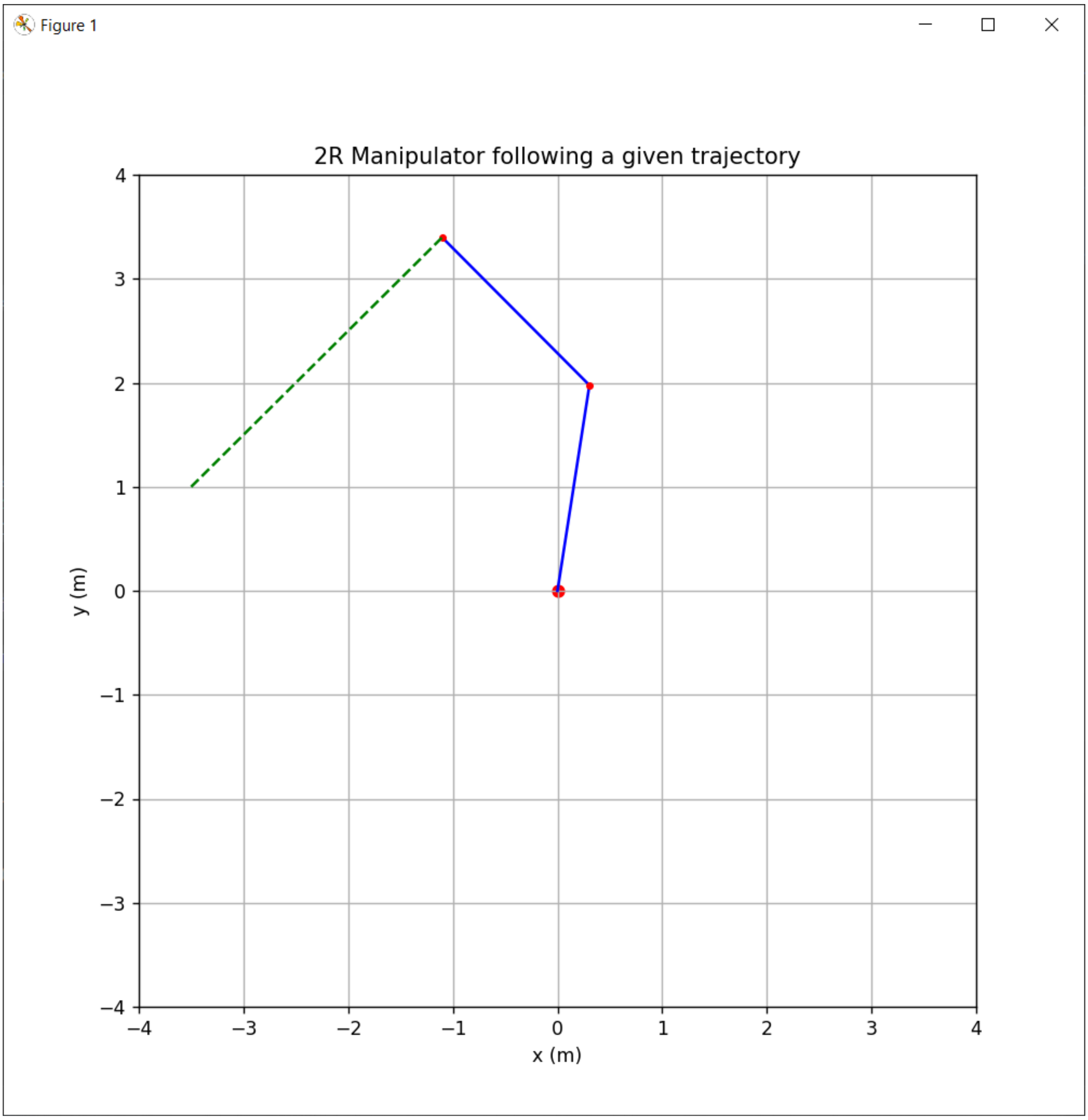


Figure 1
2R Manipulator following a given trajectory
y (m)
x (m)
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3
4

TASK 2

Source Code: (Link to GitHub)

```
import numpy as np
import math
from matplotlib import pyplot as plt
from matplotlib.animation import FuncAnimation

l1 = 2.0 # length of link 1 in m
l2 = 2.0 # length of link 2 in m
num_steps = 100000 # no. of divisions of the line of wall
num_frames = 50 # no. of frames in animation
N = 500 # normal force on wall in N

# here, angles q1 and q2 are both measured from the x-axis

# inverse kinematics
def inv_kin (x, y):
    theta = np.arccos((x**2 + y**2 - l1**2 - l2**2) / (2 * l1 * l2))
    q1 = np.arctan2(y, x) - np.arctan2((l2 * np.sin(theta)), (l1 + l2 *
np.cos(theta)))
    q2 = q1 + theta
    return q1, q2

# forward kinematics
def fwd_kin (q1, q2):
    x = l1 * np.cos(q1) + l2 * np.cos(q2)
    y = l1 * np.sin(q1) + l2 * np.sin(q2)
    return x, y

fig, ax = plt.subplots(figsize=(8, 8))  # Equal aspect ratio
ax.set_xlim(-4.0, 4.0)
ax.set_ylim(-4.0, 4.0)
ax.scatter(0, 0, c='r')

plt.xlabel('x (m)') #label
plt.ylabel('y (m)')
plt.title('2R Manipulator applying normal force on a wall')
plt.grid()

# plot the wall with intercepts at (4,0) and (0,4) which is inclined at an
angle of 135deg to x-axis
wall_x_int = (4, 0)
wall_y_int = (0, 4)
```

```
wall = plt.plot(np.array(wall_x_int), np.array(wall_y_int), 'g-', linewidth =
4)

wall_points = [(1 - i) * np.array(wall_x_int) + i * np.array(wall_y_int) for i
in np.linspace(0, 1, num_steps)]

# this loop iterates through the all the points of the wall equation and stops
when value of angle q2 = 45deg
# i.e. it stops when it finds a point where link 2 is perpendicular to the
wall
for x, y in wall_points:
        q1, q2 = inv_kin(x, y)
        if round(q2, 4) == round(np.pi / 4, 4): # rounding off up to 4 decimal
places
            break

# (x, y) is the point where the end effector touches the wall

# resolving normal force into components along the individual axes
Nx = N * np.cos(np.pi / 4)
Ny = N * np.sin(np.pi / 4)

# calculation of torques
T1 = Ny * l1 * np.cos(q1) - Nx * l1 * np.sin(q1)
T2 = Ny * l2 * np.cos(q2) - Nx * l2 * np.sin(q2) # ?????

contact_point = "Point of contact : (" + str(round(x, 2)) + ", " +
str(round(y, 2)) + ")"
app_t1 = "Torque applied on link 1 = " + str(round(T1, 3)) + " N"
app_t2 = "Torque applied on link 2 = " + str(round(T2, 3)) + " N"

# printing of data on to graph
plt.text(-3.5, 3, "Normal Force = 500 N")
plt.text(-3.5, 2.8, "Wall intercepts : (4, 0) and (0, 4)")
plt.text(-3.5, 2.6, contact_point)
plt.text(-3.5, 2.4, app_t1)
plt.text(-3.5, 2.2, app_t2)

# showing trajectory of robot till it reaches desired point on wall

# create planned trajectory
t_max = 5.0   # total simulation time (s)
start_point = (-1, -3.5) # point of start of end effector
end_point = (x, y)
trajectory = [(1 - i) * np.array(start_point) + i * np.array(end_point) for i
in np.linspace(0, 1, num_frames)]
joint_angles = [inv_kin(a, b) for a, b in trajectory]
```

```
link1, = plt.plot([], [], 'b-')
link2, = plt.plot([], [], 'b-')
point1, = plt.plot([], [], 'r.')
point2, = plt.plot([], [], 'r.')

end_effector_path, = plt.plot([], [], 'g--', label="End-Effector Path")

def animate(i):
    q1, q2 = joint_angles[i]
    a, b = fwd_kin(q1, q2)

    link1.set_data([0, l1 * np.cos(q1)], [0, l1 * np.sin(q1)])
    point1.set_data(l1 * np.cos(q1), l1 * np.sin(q1))
    link2.set_data([l1 * np.cos(q1), a], [l1 * np.sin(q1), b])
    point2.set_data(a, b)

    end_effector_path.set_data([p[0] for p in trajectory[:i+1]], [p[1] for p
in trajectory[:i+1]])

ani = FuncAnimation(fig, animate, frames = num_frames, interval = (t_max /
num_frames) * 1000, repeat = False)

plt.show()
```

Output:

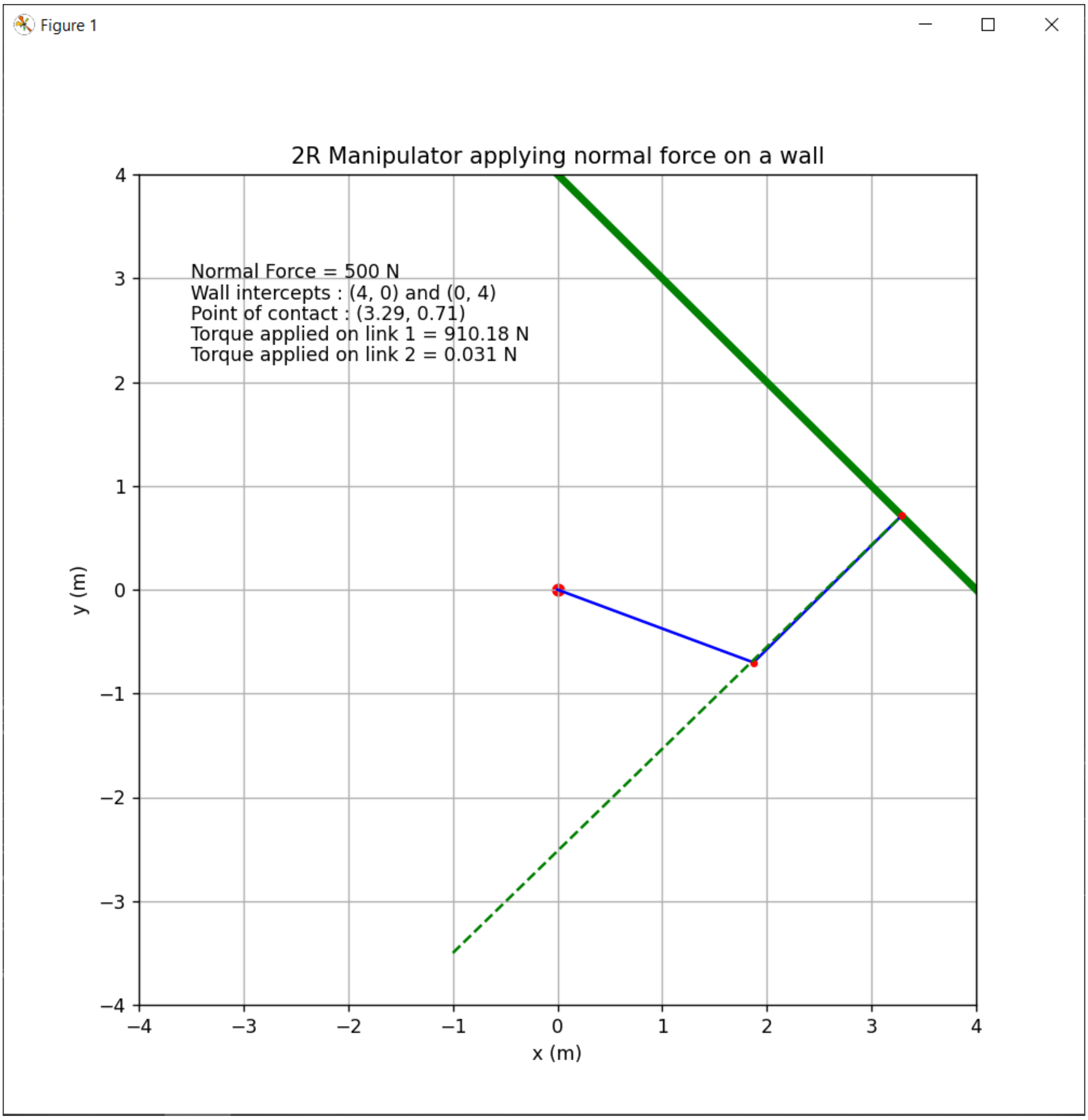


Figure 1
2R Manipulator applying normal force on a wall
Normal Force = 500 N
Wall intercepts : (4, 0) and (0, 4)
Point of contact : (3.29, 0.71)
Torque applied on link 1 = 910.18 N
Torque applied on link 2 = 0.031 N
y (m)
x (m)
4
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3
4

TASK 3

Source Code: (Link to GitHub)

```
import numpy as np
import math
from matplotlib import pyplot as plt
from matplotlib.animation import FuncAnimation

l1 = 2.0 # length of link 1 in m
l2 = 2.0 # length of link 2 in m
k = 600 # spring constant in N/m

# here, angles q1 and q2 are both measured from the x-axis

# inverse kinematics
def inv_kin (x, y):
    theta = np.arccos((x**2 + y**2 - l1**2 - l2**2) / (2 * l1 * l2))
    q1 = np.arctan2(y, x) - np.arctan2((l2 * np.sin(theta)), (l1 + l2 *
np.cos(theta)))
    q2 = q1 + theta
    return q1, q2

# forward kinematics
def fwd_kin (q1, q2):
    x = l1 * np.cos(q1) + l2 * np.cos(q2)
    y = l1 * np.sin(q1) + l2 * np.sin(q2)
    return x, y

# create planned trajectory
t_max = 5   # total simulation time (s)
num_steps = 50
num_frames = 100
mean_pos = (3, 1) # mean position of spring
ex_pos_1 = (3, 0)
ex_pos_2 = (3, 2)
traj_1 = [(1 - i) * np.array(ex_pos_1) + i * np.array(ex_pos_2) for i in
np.linspace(0, 1, num_steps)]
traj_2 = [(1 - i) * np.array(ex_pos_1) + i * np.array(ex_pos_2) for i in
np.linspace(1, 0, num_steps)]
trajectory = traj_1 + traj_2
joint_angles = [inv_kin(x, y) for x, y in trajectory]

# preparing the plot
fig, ax = plt.subplots(figsize=(8, 8))  # Equal aspect ratio
ax.set_xlim(-5.0, 5.0)
ax.set_ylim(-5.0, 5.0)
```

```
ax.scatter(x=0, y=0, c='r')

# plotting grid lines
ticks = np.arange(-5, 5, 1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)

plt.xlabel('x (m)') #label
plt.ylabel('y (m)')
plt.title('2R Manipulator acting as Spring')
plt.grid()

plt.text(3.2, 0.9, "Mean Position : (3, 1)")
plt.plot(3, 1, 'g.')
plt.text(3.2, -0.1, "Extreme Position : (3, 0)")
plt.plot(3, 0, 'g.')
plt.text(3.2, 1.9, "Extreme Position : (3, 2)")
plt.plot(3, 2, 'g.')
plt.plot([3, 3], [0, 2], 'g--')

# calculation of spring torques
def spring_torque_calc(q1, q2):
    t1s = k * (l1 * np.sin(q1) + l2 * np.sin(q2)) * l1 * np.cos(q1) - k * (l1
* np.cos(q1) + l2 * np.cos(q2)) * l1 * np.sin(q1)
    t2s = k * (l1 * np.sin(q1) + l2 * np.sin(q2)) * l2 * np.cos(q2) - k * (l1
* np.cos(q1) + l2 * np.cos(q2)) * l2 * np.sin(q2)
    return t1s, t2s

plt.text(-3.5, 3, "Spring Constant = 600 N/m")
plt.text(-3.5, 2.8, "Instantaneous spring torque on link 1
=                    Nm")
plt.text(-3.5, 2.6, "Instantaneous spring torque on link 2
=                    Nm")

link1, = plt.plot([], [], 'b-')
link2, = plt.plot([], [], 'b-')
point1, = plt.plot([], [], 'r.')
point2, = plt.plot([], [], 'r.')



def animate(i):
    q1, q2 = joint_angles[i]
    x, y = fwd_kin(q1, q2)
    t1s, t2s = spring_torque_calc(q1, q2)

    t1s_disp = plt.text(1.1, 2.8, round(t1s, 3))
    t2s_disp = plt.text(1.1, 2.6, round(t2s, 3))
```

```
    link1.set_data([0, l1 * np.cos(q1)], [0, l1 * np.sin(q1)])
    point1.set_data(l1 * np.cos(q1), l1 * np.sin(q1))
    link2.set_data([l1 * np.cos(q1), x], [l1 * np.sin(q1), y])
    point2.set_data(x, y)

    plt.pause(0.0001)
    t1s_disp.remove()
    t2s_disp.remove()

ani = FuncAnimation(fig, animate, frames=num_frames, interval=5)

plt.show()
```

Output:

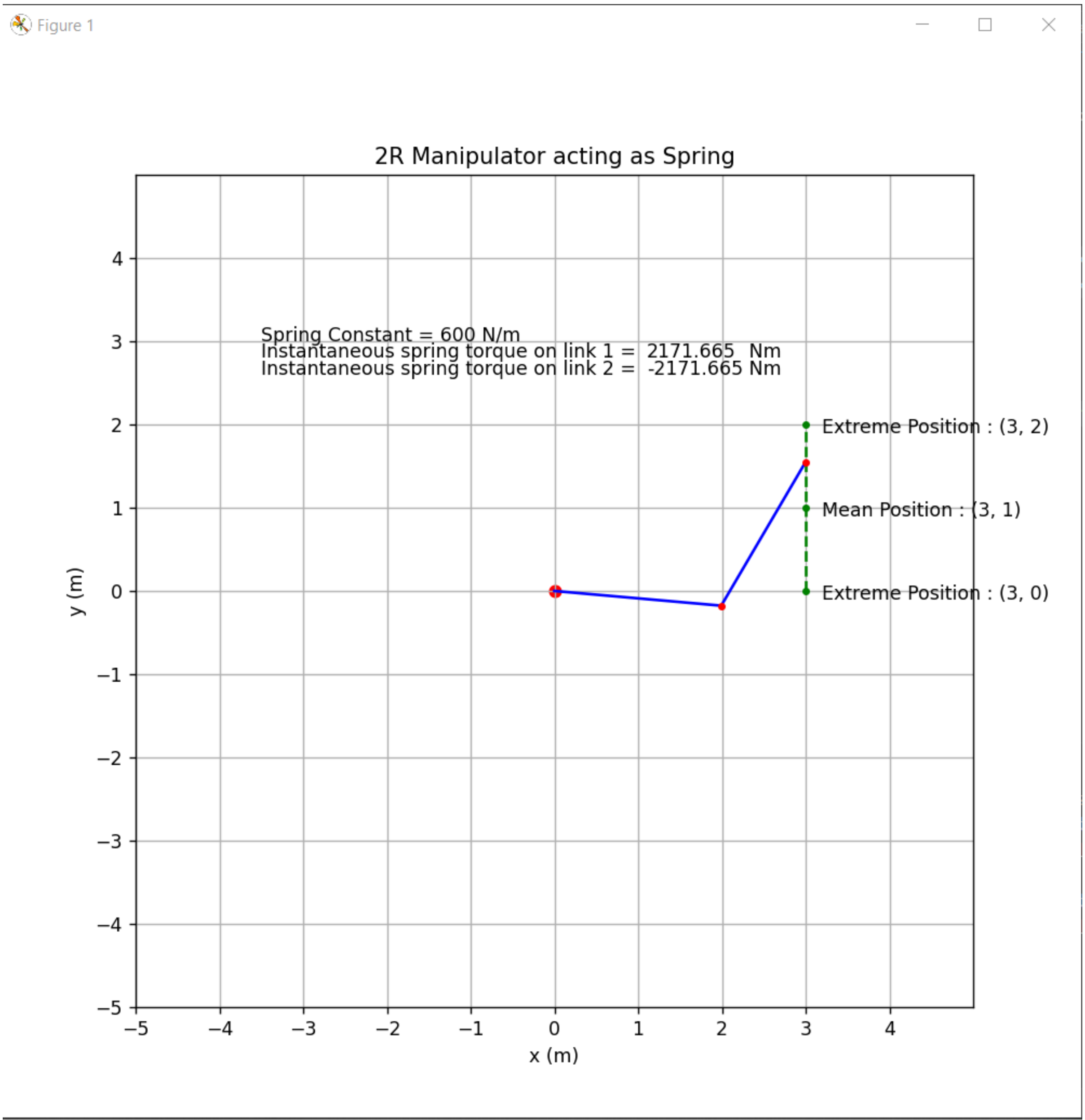


Figure 1
2R Manipulator acting as Spring
Spring Constant = 600 N/m
Instantaneous spring torque on link 1 = 2171.665 Nm
Instantaneous spring torque on link 2 = -2171.665 Nm
Extreme Position : (3, 2)
Mean Position : (3, 1)
Extreme Position : (3, 0)
y (m)
x (m)

TASK 4

Source Code: (Link to GitHub)

```
import numpy as np
import math
from matplotlib import pyplot as plt
from matplotlib.animation import FuncAnimation

l1 = 2.0 # length of link 1 in m
l2 = 2.0 # length of link 2 in m

# here, angles q1 and q2 are both measured from the x-axis

# inverse kinematics
def inv_kin (x, y):
    theta = np.arccos((x**2 + y**2 - l1**2 - l2**2) / (2 * l1 * l2))
    q1 = np.arctan2(y, x) - np.arctan2((l2 * np.sin(theta)), (l1 + l2 *
np.cos(theta)))
    q2 = q1 + theta
    return q1, q2

# forward kinematics
def fwd_kin (q1, q2):
    x = l1 * np.cos(q1) + l2 * np.cos(q2)
    y = l1 * np.sin(q1) + l2 * np.sin(q2)
    return x, y

# preparing the plot
fig, ax = plt.subplots(figsize=(8, 8))  # Equal aspect ratio
ax.set_xlim(-4.0, 4.0)
ax.set_ylim(-4.0, 4.0)

plt.xlabel('x (m)') #label
plt.ylabel('y (m)')
plt.title('Workspace of 2R Manipulator')
plt.grid()

for q1 in range(35, 145, 1):
    for q2 in range(35, 145, 1):
        x, y = fwd_kin(math.radians(q1), math.radians(q2))
        plt.scatter(x, y, s = 1, c='b')

plt.show()
```

Output:

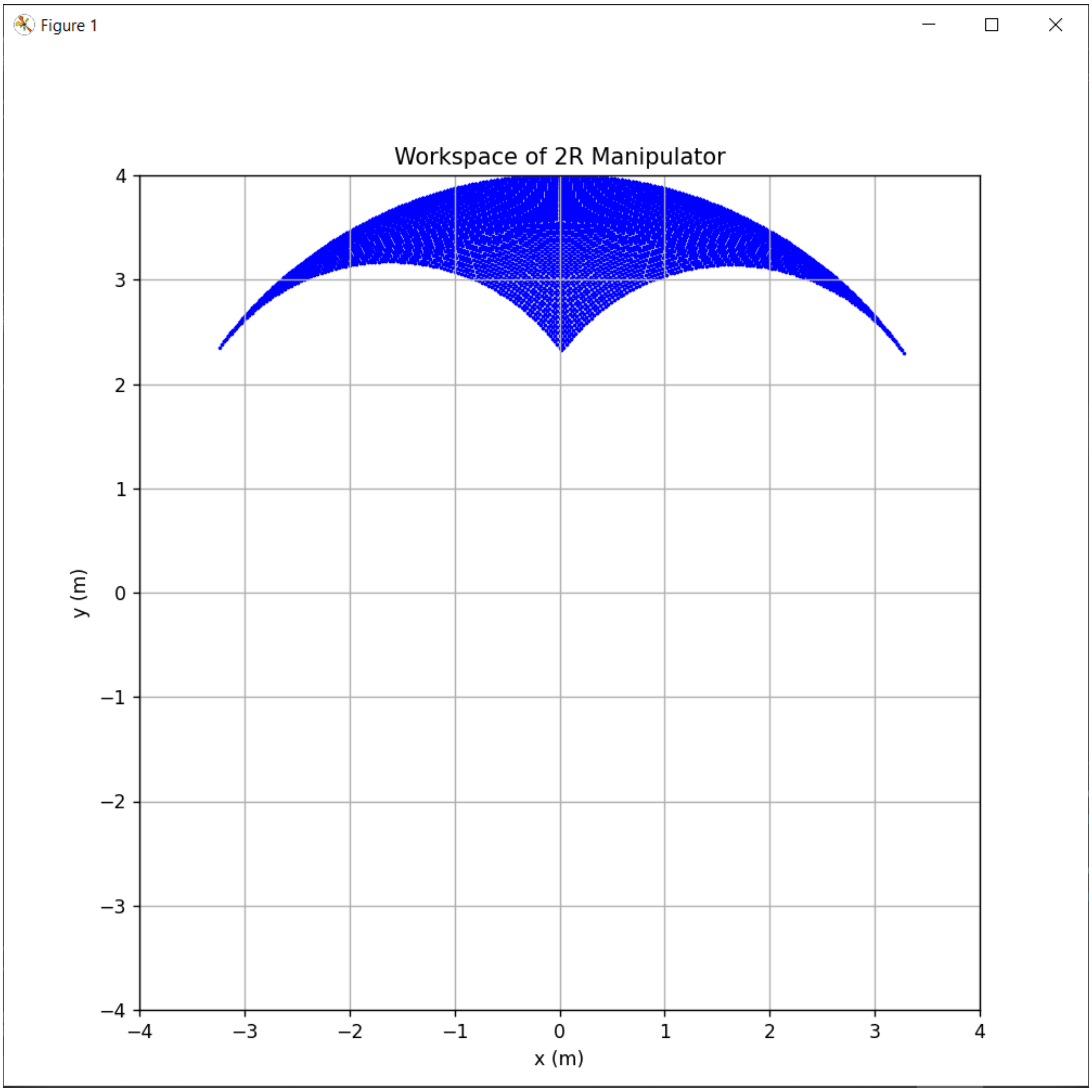


Figure 1
Workspace of 2R Manipulator
4
3
2
1
0
−1
−2
−3
−4
y (m)
−4
−3
−2
−1
0
1
2
3
4
x (m)

REFERENCES

1. Matplotlib documentation: https://matplotlib.org/stable/index.html
2. Matplotlib FuncAnimation tutorial: https://www.geeksforgeeks.org/matplotlib-animation-funcanimation-class-in-python/
3. Matplotlib Animation Tutorial: https://jakevdp.github.io/blog/2012/08/18/matplotlib-animation-tutorial/
4. SciPy Manual: https://docs.scipy.org/doc/scipy/index.html
5. ChatGPT: https://chat.openai.com/
6. How to add line based on slope and intercept in Matplotlib?: https://stackoverflow.com/questions/7941226/how-to-add-line-based-on-slope-and-intercept-in-matplotlib
7. How to truncate float values?: https://stackoverflow.com/questions/783897/how-to-truncate-float-values
8. Add Text Inside the Plot in Matplotlib: https://www.geeksforgeeks.org/add-text-inside-the-plot-in-matplotlib/
9. Update text in plot: https://stackoverflow.com/questions/68800936/update-text-in-plot
10. Change grid interval and specify tick labels: https://stackoverflow.com/questions/24943991/change-grid-interval-and-specify-tick-labels