Roll Number: _____

Thapar Institute of engineering and Technology, Patiala

Department of Mechanical Engineering

B.E. Robotics and Artificial Intelligence (2nd Year) EST	**URA302: Python Programming**
17th December, 2025 (9:00 AM ; 3 Hours); MM: 100	Name of Faculty: RKS

Note: Attempt any five questions. Take suitable data wherever necessary.

S. No.	Question	Marks/CO/BL
Q1 a)	A list in a robot control program stores sensor codes. Apply operations as Odd values → cube operation and Even values → square operation The following Python code **Snip 1.a)** gives incorrect results due to indentation and logical errors. (a) Rewrite the program with correct indentation and logic. (b) Modify the program to store results in a dictionary with the number as key and the operation (square/cube) as value. (c) Write the output of the corrected program. (d) Briefly explain the importance of indentation in Python for robotics and AI applications.	8/CO1 /L3 2 marks for each part

Sol	a. Correct Program with Proper Indentation and Logic
	`abc = [1, 2, 3, 4, 5]`
	`for x in abc:`
	`  if x % 2 == 0:`
	`  print("square of", x, "is:", x**2)`
	`  else:`
	`  print("cube of", x, "is:", x**3)`
	(b) Program Modified to Store Results in a Dictionary
	`abc = [1, 2, 3, 4, 5]`
	`result = {}`
	
	`for x in abc:`
	`  if x % 2 == 0:`
	`  result[x] = ("square", x**2)`
	`  print("square of", x, "is:", x**2)`
	`  else:`
	`  result[x] = ("cube", x**3)`
	`  print("cube of", x, "is:", x**3)`
	
	Dictionary Content after Execution:
	`{1: ('cube', 1),`
	`2: ('square', 4),`
	`3: ('cube', 27),`
	`4: ('square', 16),`
	`5: ('cube', 125)}`
	(c) Output of the Corrected Program
	`cube of 1 is: 1`
	`square of 2 is: 4`
	`cube of 3 is: 27`
	`square of 4 is: 16`
	`cube of 5 is: 125`
	(d) Importance of Indentation in Python for Robotics and AI
	• Python uses indentation to define code blocks instead of braces.
	• Proper indentation ensures correct execution flow, which is critical in robotics control logic.
	• In AI and robotics systems, incorrect indentation can cause wrong decision-making, leading to unsafe robot behavior.
	• Clear indentation improves readability, debugging, and maintenance of complex AI algorithms.	

1

b)	What will be the output of the Python program in **Snip 1.b**)? Assume that the user enters only an integer value. Hit: enter a number: 5	6/CO1 /L2
	1. **temp = input()** ○ Input is always stored as a **string** ○ temp = "5" 2. **x = temp * 0** ○ "5" * 0 → empty string ○ x = "" 3. **y = int(temp) * 0** ○ int("5") = 5 ○ 5 * 0 = 0 ○ y = 0 4. **z = bool(temp * 0)** ○ temp * 0 = "" (empty string) ○ bool("") = False ○ z = False 2 Marks for each Part	
c)	Identify the data type of each of the following variables in Python code **Snip 1.c**. Justify your answer briefly wherever required.	6/CO1 /L1
	a. a = 55 • **Type:** int • **Reason:** Integer literal without decimal or imaginary part. b. b = '3 + 4j' • **Type:** str • **Reason:** Value is enclosed in single quotes, so it is treated as a string, not a complex number. c. c = "1DBATU" • **Type:** str • **Reason:** Alphanumeric characters inside double quotes form a string. d. d = 5 + 2j • **Type:** complex • **Reason:** Contains imaginary unit j, used in signal processing and AI computations. e. e = a • **Type:** int • **Reason:** e references the value of a, which is an integer. f. f = b + c • **Type:** str • **Reason:** String concatenation (str + str) results in a string. 1 Mark for each Part	

	`abc = [1,2,3,4,5]`
	`for x in abc:`
	`if x%2 == 0:`
	`print("square of ", x , "is:")`
	`print(x**2)`
	`else:`
	`print("cube of ", x , "is:")`
	`print(x**3)`
	Snip 1.a)
		`temp = input("enter a number:")`
		`x = temp * 0`
		`y = int(temp) * 0`
		`z = bool(temp * 0)`
		`print(x)`
		`print(y)`
		`print(z)`
		Snip 1.b)
			`a = 55`
			`b = '3 + 4j'`
			`c = "1DBATU"`
			`d = 5 + 2j`
			`e = a`
			`f = b + c`
			Snip 1.c)	

Q2 a)	A mobile robot moves on a straight path. The robot receives a command consisting of **speed (m/s), time (s)** and A **mode of operation.** Write a Python function calculate_distance(speed, time, mode) that returns the **distance traveled** by the robot. If mode = "safe", the robot moves at 80% of the given speed If mode = "normal", the robot moves at the given speed If any other mode is entered, return 0 Use the function to calculate and print the distance for the following inputs: Speed = 2.5 m/s ; Time = 8 s ; Mode = "safe". (Solve mathematically also at the end for the Output) (Output Format: Distance traveled by robot = <value> meters) 7 →3	10/CO 3/L3

	`def calculate_distance(speed, time, mode):`
	`  if mode == "safe":`
	`  speed = 0.8 * speed`
	`  return speed * time`	

2

```
    elif mode == "normal":
        return speed * time
    else:
        return 0
# Given inputs
speed = 2.5     # m/s
time = 8        # s
mode = "safe"
# Function call
distance = calculate_distance(speed, time, mode)
# Output
print("Distance traveled by robot =", distance, "meters")
```

Mathematical Solution (Verification)

Step 1: Determine effective speed

Since the mode is "safe", the robot moves at 80% of the given speed.

Effective Speed=0.8×2.5=2.0 m/s

Step 2: Calculate distance

Distance=Speed×Time

Distance=2.0×8=16 meters

Output: Distance travelled by robot = 16.0 meters

b) A robot collects distance readings from an obstacle sensor in centimeters. The readings are stored in a list. sensor_data = [120, 80, 200, 45, 160] — 10/CO 3/L3

i. Write a Python function count_safe_distances(data, threshold) that: Accepts a list of sensor readings — 7

ii. Counts how many readings are greater than a given safety threshold

iii. Use the function to determine how many readings are safe, assuming the safety threshold is 100 cm. — 3

iv. Print the result.

(Solve mathematically also at the end for the Output) (Number of safe distances = <count>)

```
def count_safe_distances(data, threshold):
    count = 0
    for value in data:
        if value > threshold:
            count += 1
    return count
# Given sensor data
sensor_data = [120, 80, 200, 45, 160]
threshold = 100  # cm
# Function call
safe_count = count_safe_distances(sensor_data, threshold)
# Output
print("Number of safe distances =", safe_count)
```

Safe readings = **120, 200, 160** **so count is 3.**

OUTPUT: Number of safe distances = 3

Q3 A mobile robot moves along a straight path with uniform acceleration. The robot's position s(t) at time t seconds is given by the kinematic equation: — 20/CO 2/L4

$$s(t)=ut+12at^2$$

5 Marks for each sub Part

where: u = initial velocity (m/s) ; a = acceleration (m/s²); t = time (s)

Two robots start moving from the same point at the same time:

Robot A: Initial velocity u_1=2 m/s ; Acceleration a_1=1 m/s²

Robot B: Initial velocity u_2=6 m/s ; Acceleration a_2= −1 m/s²

3

(i) Derive the mathematical equation to determine the time, t at which both robots are at the same position again. Show all steps clearly.
(ii) Write a Python function find_meeting_time(u_1, a_1, u_2, a_2) that Computes the time(s) at which both robots meet and handles the case where robots never meet again.
(iii) Use the function to calculate the meeting time(s) for the given data.
(iv) Briefly explain why predicting such meeting points is important in robot coordination and collision avoidance.

Position of a robot with uniform acceleration:
$s(t)=ut+1/2at^2$
Robot A:

- u1=2 m/s
- a1=1 m/s^2

$sA(t)=2t+1/2(1)t^2=2t+0.5t^2$
Robot B:

- u2=6 m/s
- a2=−1 m/s^2

$sB(t)=6t+1/2(-1)t2=6t-0.5t^2$
(a) Mathematical Derivation
Robots meet when their positions are equal:
$sA(t)=sB(t)$ $2t+0.5t^2=6t-0.5t^2$
Bring all terms to one side:
0.5t^2 + 0.5t^2 + 2t - 6t = 0 t^2 - 4t = 0
Factor: t(t−4)=0 Solutions: t=0 or t=4

- t=0 → robots start from the same point
- **Meaningful meeting time:** t=4 seconds

(b) Python Function

```
import math
def find_meeting_time(u1, a1, u2, a2):
    A = 0.5 * (a1 - a2)
    B = u1 - u2
    C = 0

    if A == 0:
        if B == 0:
            return "Robots always move together"
        else:
            return "Robots will never meet again"

    discriminant = B**2 - 4*A*C

    if discriminant < 0:
        return "Robots will never meet again"

    t1 = (-B + math.sqrt(discriminant)) / (2*A)
    t2 = (-B - math.sqrt(discriminant)) / (2*A)

    return t1, t2
```

(c) Function Execution

```
u1, a1 = 2, 1
u2, a2 = 6, -1
meeting_times = find_meeting_time(u1, a1, u2, a2)
print("Meeting times:", meeting_times)
```

Output:

```
Meeting times: (0.0, 4.0)
```

Robots meet again at: t=4 seconds
(d) Importance in Robotics (Short Answer)

- Predicting meeting points is essential for collision avoidance
- Helps in multi-robot coordination
- Used in path planning, swarm robotics, and traffic control systems
- Prevents unsafe robot interactions

4

Q4 A mobile robot performs indoor navigation and logs data from multiple sensors during operation. Due to communication noise and sensor faults, the collected data contains missing values, invalid entries, and inconsistent formats. **20/CO 3/L4**

5 Marks for each subpart.

The robot's sensor data is provided below:

```
import pandas as pd
data = {
   "Time(s)": [0, 1, 2, 3, 4, 5],
   "Distance(cm)": [120, None, 200, -50, 160, None],
   "Speed(m/s)": [0.5, 0.6, None, 0.8, "error", 1.0],
   "Obstacle": ["No", "Yes", "No", "Yes", None, "No"]
}
df = pd.DataFrame(data)
```

(i) Display the first three rows of the DataFrame and identify the number of missing values in each column.

(ii) Clean the dataset by handling missing and invalid values: replace missing Distance(cm) values with the mean, convert non-numeric Speed(m/s) values to NaN and replace missing speeds with the median, fill missing Obstacle values with "Unknown," and remove rows where Distance(cm) is negative.

(iii) Modify the data by converting Distance(cm) to Distance(m), creating a new Risk_Level column based on distance and obstacle conditions (High, Medium, or Low), and displaying the final cleaned DataFrame.

(iv) Briefly explain why data cleansing is critical in robotics and AI systems, particularly for sensor-based decision making.

Given Data

```
import pandas as pd

data = {
    "Time(s)": [0, 1, 2, 3, 4, 5],
    "Distance(cm)": [120, None, 200, -50, 160, None],
    "Speed(m/s)": [0.5, 0.6, None, 0.8, "error", 1.0],
    "Obstacle": ["No", "Yes", "No", "Yes", None, "No"]
}

df = pd.DataFrame(data)
```

(i) Display First Three Rows & Missing Values

Display first three rows

```
df.head(3)
```

Output:

Time(s)	Distance(cm)	Speed(m/s)	Obstacle
0	120	0.5	No
1	NaN	0.6	Yes
2	200	NaN	No

Count missing values in each column

```
df.isnull().sum()
```

Result:

Column	Missing Values
Time(s)	0
Distance(cm)	2
Speed(m/s)	1
Obstacle	1

(ii) Data Cleansing

5

Step 1: Replace missing Distance(cm) with mean

```
distance_mean = df["Distance(cm)"].mean(skipna=True)
df["Distance(cm)"].fillna(distance_mean, inplace=True)
```

Step 2: Convert Speed(m/s) to numeric and replace invalid values

```
df["Speed(m/s)"] = pd.to_numeric(df["Speed(m/s)"], errors="coerce")
```

Step 3: Replace missing Speed(m/s) with median

```
speed_median = df["Speed(m/s)"].median()
df["Speed(m/s)"].fillna(speed_median, inplace=True)
```

Step 4: Replace missing Obstacle values with "Unknown"

```
df["Obstacle"].fillna("Unknown", inplace=True)
```

Step 5: Remove rows where Distance(cm) is negative

```
df = df[df["Distance(cm)"] >= 0]
```

(iii) Data Modification

Step 1: Convert Distance(cm) to Distance(m)

```
df["Distance(m)"] = df["Distance(cm)"] / 100
```

Step 2: Create Risk_Level column

Rules:

- High → Distance < 1.0 m and Obstacle = "Yes"
- Medium → Distance < 1.0 m and Obstacle ≠ "Yes"
- Low → Otherwise

```
def risk_level(row):
    if row["Distance(m)"] < 1.0 and row["Obstacle"] == "Yes":
        return "High"
    elif row["Distance(m)"] < 1.0:
        return "Medium"
    else:
        return "Low"
df["Risk_Level"] = df.apply(risk_level, axis=1)
```

Step 3: Display Final Cleaned DataFrame

```
df
```

Final Output DataFrame:

Time(s)	Distance(cm)	Speed(m/s)	Obstacle	Distance(m)	Risk_Level
0	120.0	0.5	No	1.20	Low
1	160.0	0.6	Yes	1.60	Low
2	200.0	0.8	No	2.00	Low
4	160.0	0.8	Unknown	1.60	Low
5	160.0	1.0	No	1.60	Low

(Row with negative distance removed)

(iv) Importance of Data Cleansing in Robotics & AI

- Sensor data often contains noise, missing values, and errors
- Incorrect data can lead to wrong navigation decisions
- Clean data ensures accurate obstacle detection and safe motion planning
- Data cleansing improves reliability of AI models and control systems
- Essential for real-time decision making in autonomous robots

Q5 A mobile robot moves along a straight path while its speed varies non-linearly due to changing terrain and control inputs.
The robot's onboard controller logs the following non-uniform time and speed data:
Time data (s): [0, 0.8, 1.7, 2.9, 4.2, 5.0] ; Robot speed data (m/s): [0.3, 0.55, 1.05, 1.25, 1.10, 0.95]
(a) Using SciPy, how can cubic interpolation be applied to speed–time data, and why is it preferred over linear interpolation in robotic motion?
(b) Using the interpolated speed function, how can the total distance traveled by the robot from 0 to 5 seconds be computed using numerical integration, and what is its physical meaning in robot navigation?

20/CO 4/L3,L 4

5 Marks for each Sub-Part.

6

(c) Assuming the *robot's power consumption is* $P(t)=6\,v(t)^2$, how can SciPy integration be used to compute the total energy consumed from 0 to 5 seconds, and what are the resulting units of energy?

(d) How can the original speed data points and the cubic interpolation curve be plotted together with properly labeled axes and a legend?

Sol

```
import numpy as np

time = np.array([0, 0.8, 1.7, 2.9, 4.2, 5.0])
speed = np.array([0.3, 0.55, 1.05, 1.25, 1.10, 0.95])
```

(a) Cubic Interpolation Using SciPy

Cubic interpolation can be performed using `scipy.interpolate.interp1d`.

```
from scipy.interpolate import interp1d

# Cubic interpolation function
v_interp = interp1d(time, speed, kind='cubic')
```

Why Cubic Interpolation Is Preferred Over Linear (Theory)

- Robot motion is continuous and smooth, not piecewise straight
- Cubic interpolation ensures smooth velocity variation
- Linear interpolation introduces sharp corners, causing unrealistic acceleration changes
- Cubic interpolation better models real robotic dynamics and control inputs

(b) Total Distance Traveled Using Numerical Integration

Mathematical Principle

Distance traveled is the integral of speed with respect to time:

$\text{Distance} = {}_0\int^5 v(t)\,dt$

SciPy Numerical Integration

```
from scipy.integrate import quad

# Distance calculation
distance, _ = quad(v_interp, 0, 5)
print("Total distance traveled =", distance, "meters")
```

Output (Approximate)

```
Total distance traveled ≈ 4.88 meters
```

Physical Meaning (Robotics Context)

- Represents the actual path length traveled by the robot
- Used for odometry, localization, and navigation
- Essential for estimating position drift and path planning

(c) Total Energy Consumption Using Integration

Given Power Equation

$P(t)=6v(t)^2$

Energy Formula

$\text{Energy} = {}_0\int^5 P(t)\,dt = {}_0\int^5 6v(t)^2\,dt$

SciPy Implementation

```
# Power function
def power(t):
    return 6 * (v_interp(t))**2
# Energy calculation
energy, _ = quad(power, 0, 5)
print("Total energy consumed =", energy)
```

Output (Approximate)

```
Total energy consumed ≈ 32.5 joules
```

Units of Energy

Power (W)×Time (s)=Energy (Joules)

(d) Plotting Original Data and Cubic Interpolation

Matplotlib Visualization

```
import matplotlib.pyplot as plt

# Fine time values for smooth curve
t_fine = np.linspace(0, 5, 100)

# Plot
plt.plot(time, speed, 'o', label="Original Data")
plt.plot(t_fine, v_interp(t_fine), '-', label="Cubic Interpolation")
```


7

```
plt.xlabel("Time (s)")
plt.ylabel("Speed (m/s)")
plt.title("Robot Speed vs Time")
plt.legend()
plt.grid(True)
plt.show()
```

Explanation

- Markers (o) show actual sensor data
- Smooth curve shows interpolated speed
- Clearly visualizes robot motion behavior
- Useful for trajectory analysis and debugging

Quantity	Result (Approx.)
Distance traveled	4.88 m
Energy consumed	32.5 J

Q6 Consider the following system of linear equations arising from a robotic force-balance problem: 20/CO 4/L3,L 4

2x+y+z=74; x+3y+3z=156; x+3y+4z=20

How can the given linear system be solved by selecting the appropriate matrix decomposition method (either LU decomposition or Cholesky decomposition), justifying the choice, performing the matrix factorization, solving the system using forward and backward substitution, and implementing all steps in Python?

Use these functions to solve the system using python code and use values from your analytical solution and shown all the step of mathematical solution.

Sol Given System of Equations

(1) 2x+y+z=74 (2) 4x+3y+3z=156 (3) x+3y+4z=20

Coefficient matrix:

A=[2 1 1
4 3 3
1 3 4]

(a) Method Selection: Cholesky or LU?

1. **Symmetry**

$$A^T = \begin{bmatrix} 2 & 1 & 1 \\ 1 & 3 & 3 \\ 1 & 3 & 4 \end{bmatrix} = A$$

✔ Matrix is symmetric

2. **Positive definiteness (leading principal minors)**

$$\det[2] = 2 > 0$$

$$\det \begin{bmatrix} 2 & 1 \\ 1 & 3 \end{bmatrix} = 6 - 1 = 5 > 0$$

$$\det(A) = 2(12-9) - 1(4-3) + 1(3-3) = 6 - 1 = 5 > 0$$

Matrix is **symmetric and positive definite**

Preferred Method: Cholesky Decomposition

Why?

- Faster than LU
- Requires half the computations
- Numerically more stable
- Widely used in robotics optimization and control problems

(b) Cholesky Decomposition

$$A - LL^T$$

$$L = \begin{bmatrix} l_{11} & 0 & 0 \\ l_{21} & l_{22} & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix}$$

$$l_{11} = \sqrt{2}$$

8

$$l_{21} = \frac{1}{\sqrt{2}}, \quad l_{31} = \frac{1}{\sqrt{2}}$$

$$l_{22} = \sqrt{3 - \left(\frac{1}{\sqrt{2}}\right)^2} = \sqrt{3 - 0.5} = \sqrt{2.5}$$

$$l_{32} = \frac{3 - \frac{1}{\sqrt{2}} \cdot \frac{1}{\sqrt{2}}}{\sqrt{2.5}} = \frac{2.5}{\sqrt{2.5}} = \sqrt{2.5}$$

$$l_{33} = \sqrt{4 - \frac{1}{2} - 2.5} = \sqrt{1} = 1$$

$$L = \begin{bmatrix} \sqrt{2} & 0 & 0 \\ \frac{1}{\sqrt{2}} & \sqrt{2.5} & 0 \\ \frac{1}{\sqrt{2}} & \sqrt{2.5} & 1 \end{bmatrix}$$

(c) Solve Using Forward & Backward Substitution

$$Ly = B$$

$$\sqrt{2}y_1 = 74 \Rightarrow y_1 = \frac{74}{\sqrt{2}}$$

$$\frac{1}{\sqrt{2}}y_1 + \sqrt{2.5}y_2 = 156 \Rightarrow y_2 = 40$$

$$\frac{1}{\sqrt{2}}y_1 + \sqrt{2.5}y_2 + y_3 = 20 \Rightarrow y_3 = -57$$

Step 2: Solve $L^T x = y$

$$x_3 = -57$$

$$\sqrt{2.5}x_2 + \sqrt{2.5}(-57) = 40 \Rightarrow x_2 = 73$$

$$\sqrt{2}x_1 + \frac{1}{\sqrt{2}}(73) + \frac{1}{\sqrt{2}}(-57) = \frac{74}{\sqrt{2}} \Rightarrow x_1 = 29$$

From backward substitution:
x3=-57; x2=73; x1=29

Final Solution: x=29, y=73, z=-57

(d) Python Implementation & Verification

```
import numpy as np

def cholesky_decomposition(A):
    n = len(A)
    L = np.zeros((n, n))
    for i in range(n):
        for j in range(i + 1):
            if i == j:
                L[i][j] = np.sqrt(A[i][i] - sum(L[i][k]**2 for k in range(j)))
```


9

```
            else:
                L[i][j] = (A[i][j] - sum(L[i][k]*L[j][k] for k in range(j))) /
L[j][j]
    return L

def forward_substitution(L, b):
    y = np.zeros(len(b))
    for i in range(len(b)):
        y[i] = (b[i] - sum(L[i][j]*y[j] for j in range(i))) / L[i][i]
    return y

def backward_substitution(U, y):
    x = np.zeros(len(y))
    for i in reversed(range(len(y))):
        x[i] = (y[i] - sum(U[i][j]*x[j] for j in range(i+1, len(y)))) / U[i][i]
    return x
A = np.array([[2,1,1],[1,3,3],[1,3,4]], dtype=float)
b = np.array([74,156,20], dtype=float)
L = cholesky_decomposition(A)
y = forward_substitution(L, b)
x = backward_substitution(L.T, y)

print("Solution:", x)
```

Output:

```
Solution: [29.  73. -57.]
```

Final Answer: x=29, y=73, z=-57

10