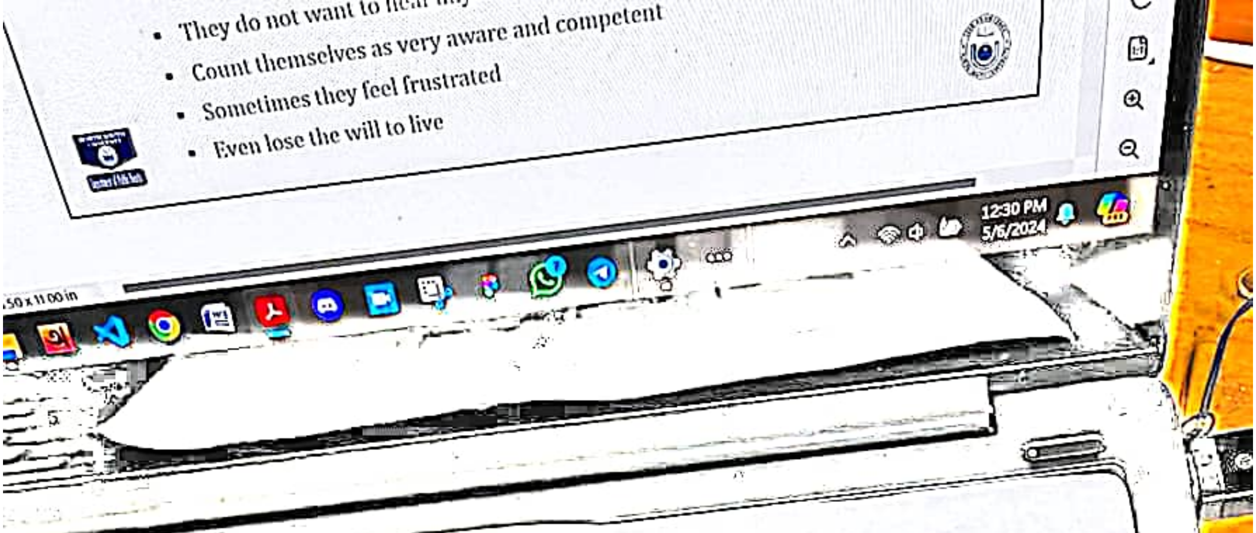


- They do not want to hear
- Count themselves as very aware and competent
- Sometimes they feel frustrated
- Even lose the will to live

S. Siddique

CSE495

Final
November 17, 2023

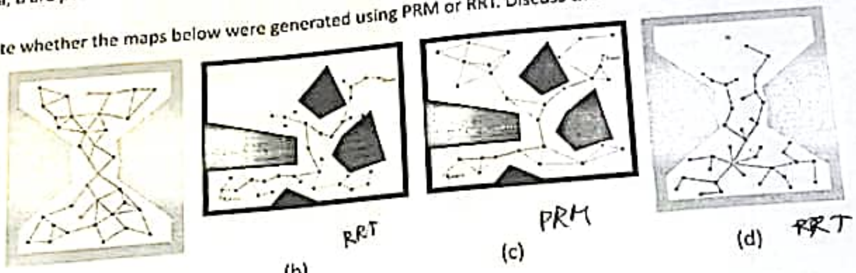
1) Consider the system:

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = -a\sin x_1 \cos x_1 - bx_2$$

Show the system is stable for the Lyapunov function $V = \frac{a}{2}\sin^2 x_1 + \frac{1}{2}x_2^2$

Here a, b are positive constants.

2) Indicate whether the maps below were generated using PRM or RRT. Discuss the reasoning for your choice.



(a) PRM (b) RRT (c) PRM (d) RRT

3) A. Describe step by step the A* grid-based search motion planning method.
B. Discuss the pros and cons of grid-based search and sampling-based motion planning methods.

4) Given an input image, describe an algorithm to implement an edge detector. If possible, use mathematical equations to describe the steps.

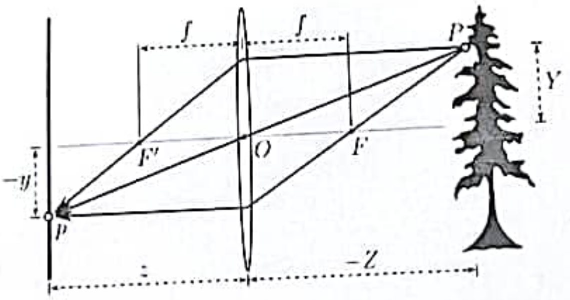
5) To solve the problem below use the following properties of matrices.

Property 1: If the eigenvalues of a matrix are negative then the system is stable and thus controllable.
Property 2: For an upper triangular matrix, the eigenvalues are the diagonal elements of the matrix.

Consider the following state-space equation for a robot: $\begin{bmatrix}\dot{\rho}\\ \dot{\alpha}\\ \dot{\beta}\end{bmatrix} = \begin{bmatrix}1 & 0\\ \alpha/\rho & 1\\ 0 & 1\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix}$

You decide to use the control law: $v = k_1\rho, \omega = k_2\alpha + k_3\beta$ where k_1, k_2, k_3 are constants.

What are the requirements on k_1, k_2, k_3 for this system to be stable?

6) 

Consider the image on the left. The point P is a point on the tree. There is a thin lens at point O with focal length f. The point p is a projection of point P on the image plane.

Using the laws of light refraction, derive the thin lens equation:

$$\frac{1}{z} + \frac{1}{Z} = \frac{1}{f}$$

y

To derive the thin lens equation $\frac{1}{z} + \frac{1}{Z} = \frac{1}{f}$, let us analyze the provided diagram and apply the principles of light refraction and geometry.

Explanation and Derivation

Key Notations:

- **Focal length** (f): Distance from the lens center (O) to the focal points (F and F').
- **Object distance** (z): Distance from the object point (P) to the lens center (O).
- **Image distance** (Z): Distance from the image point (p) to the lens center (O).
- **Object height** (Y): Height of the object (P).
- **Image height** (y): Height of the image (p).

Geometrical Setup:

- Light rays emanating from point P on the object pass through the lens and converge to the corresponding point p on the image plane.
- A parallel ray from P refracts through the lens and passes through the focal point F'.
- A ray passing through the lens center O continues undeflected.

We need to combine these observations to derive the relationship between z, Z, and f.

Step 1: Similar Triangles in the Setup

The setup forms two pairs of similar triangles:

1. **Triangle** PFO is similar to Triangle $F'pO$.
2. **Triangle** POO' (with O' as a virtual extension) is similar to Triangle $O'pO$.

Using the geometry of the first similarity:

$$\frac{Y}{z} = \frac{y}{f}.$$

Using the geometry of the first similarity:

$$\frac{Y}{z} = \frac{y}{f}.$$

Rearranging this:

$$y = \frac{Y \cdot f}{z}.$$

Using the geometry of the second similarity:

$$\frac{y}{f} = \frac{Y}{Z}.$$

Rearranging this:

$$y = \frac{Y \cdot Z}{f}.$$

Step 2: Relating Object and Image Distances

Equating the two expressions for y:

$$\frac{Y \cdot f}{z} = \frac{Y \cdot Z}{f}.$$

Canceling Y (nonzero) and rearranging:

$$\frac{1}{z} + \frac{1}{Z} = \frac{1}{f}.$$

Thin Lens Equation:

The derived equation:

$$\frac{1}{z} + \frac{1}{Z} = \frac{1}{f},$$

shows the relationship between the object distance (z), the image distance (Z), and the focal length (f) of a thin lens.

The derived equation:

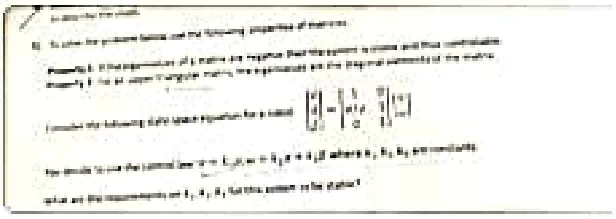
$$\frac{1}{z}+\frac{1}{Z}=\frac{1}{f},$$

shows the relationship between the object distance (z), the image distance (Z), and the focal length (f) of a thin lens.

Physical Interpretation:

1. Object Distance (z): If the object moves farther from the lens, z increases, causing the image distance (Z) to decrease.
2. Focal Length (f): A property of the lens determined by its curvature and refractive index.
3. Image Distance (Z): The position where rays converge to form the image of the object.

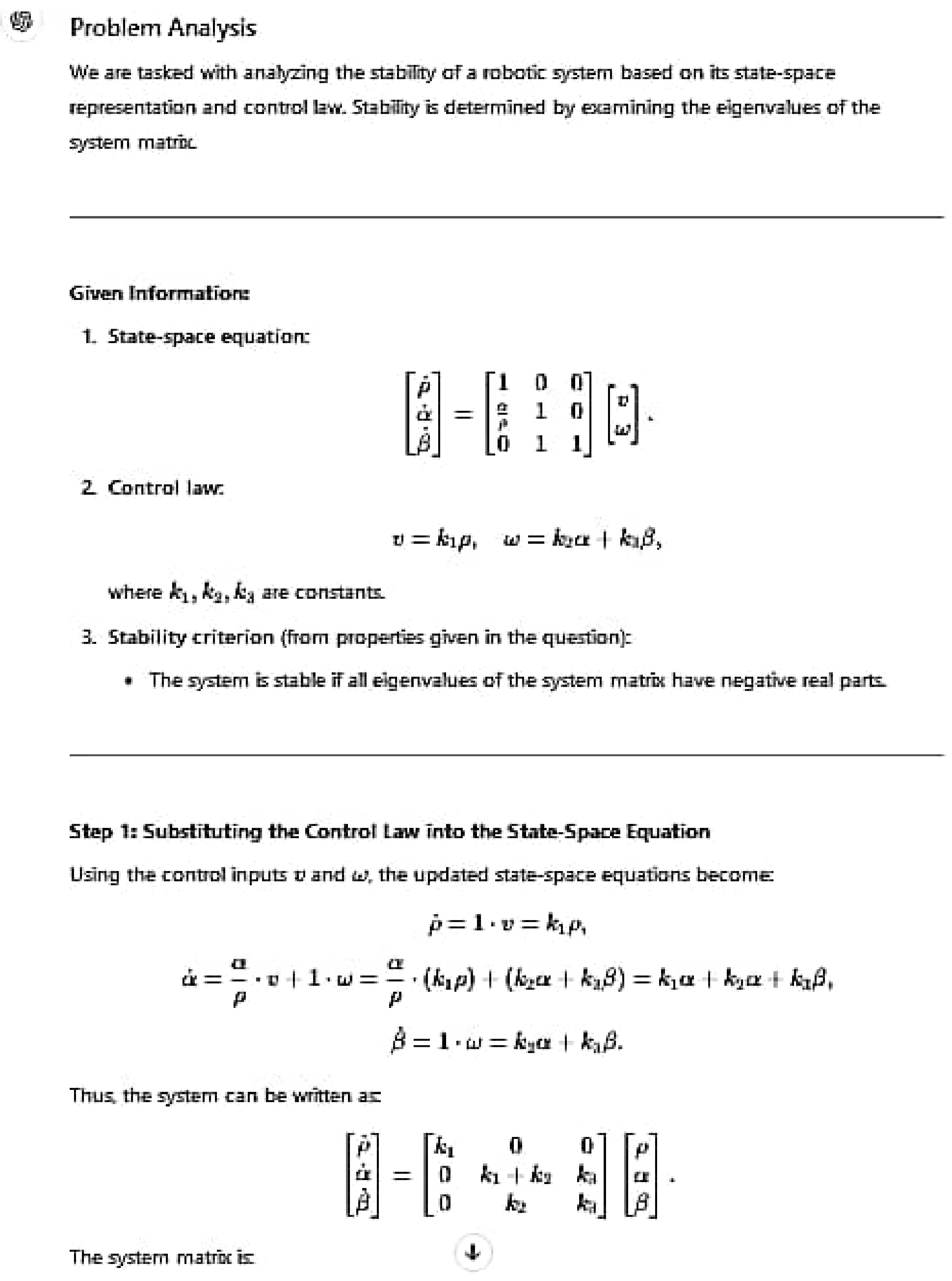
This equation is fundamental in optics and is widely used in robotics applications involving cameras, such as in the Principles of Robot Autonomy course, where cameras are modeled using the thin lens approximation.



Answer the question 5 with full description and explain elaborately everything.(PrinciplesofRobotAutonomy)

Problem Analysis

We are tasked with analyzing the stability of a robotic system based on its state-space representation and control law. Stability is determined by examining the eigenvalues of the system matrix.



Problem Analysis

We are tasked with analyzing the stability of a robotic system based on its state-space representation and control law. Stability is determined by examining the eigenvalues of the system matrix.

Given Information:

1. **State-space equation:**

$$\begin{bmatrix}\dot{\rho}\\ \dot{\alpha}\\ \dot{\beta}\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ \frac{\alpha}{\rho} & 1 & 0\\ 0 & 1 & 1\end{bmatrix}\begin{bmatrix}v\\ \omega\end{bmatrix}.$$

2. **Control law:**

$$v = k_1\rho, \quad \omega = k_2\alpha + k_3\beta,$$

   where k_1, k_2, k_3 are constants.

3. **Stability criterion (from properties given in the question):**
   - The system is stable if all eigenvalues of the system matrix have negative real parts.

Step 1: Substituting the Control Law into the State-Space Equation

Using the control inputs v and ω, the updated state-space equations become:

$$\dot{\rho} = 1\cdot v = k_1\rho,$$

$$\dot{\alpha} = \frac{\alpha}{\rho}\cdot v + 1\cdot\omega = \frac{\alpha}{\rho}\cdot(k_1\rho) + (k_2\alpha + k_3\beta) = k_1\alpha + k_2\alpha + k_3\beta,$$

$$\dot{\beta} = 1\cdot\omega = k_2\alpha + k_3\beta.$$

Thus, the system can be written as:

$$\begin{bmatrix}\dot{\rho}\\ \dot{\alpha}\\ \dot{\beta}\end{bmatrix} = \begin{bmatrix}k_1 & 0 & 0\\ 0 & k_1 + k_2 & k_3\\ 0 & k_2 & k_3\end{bmatrix}\begin{bmatrix}\rho\\ \alpha\\ \beta\end{bmatrix}.$$

The system matrix is:

$$\lfloor \beta \rfloor \quad \lfloor 0 \quad k_2 \quad k_3 \rfloor \lfloor \beta \rfloor$$

The system matrix is:

$$A = \begin{bmatrix} k_1 & 0 & 0 \\ 0 & k_1 + k_2 & k_3 \\ 0 & k_2 & k_3 \end{bmatrix}.$$

Step 2: Eigenvalues of the System Matrix

The eigenvalues of matrix A determine the stability of the system.

1. The first eigenvalue is directly given by k_1 (from the top-left diagonal entry).
2. The remaining eigenvalues are obtained by solving the characteristic equation for the 2×2 submatrix:

   $$B = \begin{bmatrix} k_1 + k_2 & k_3 \\ k_2 & k_3 \end{bmatrix}.$$

   The characteristic equation is:

   $$\det(\lambda I - B) = 0,$$

   which expands to:

   $$\begin{vmatrix} \lambda - (k_1 + k_2) & -k_3 \\ -k_2 & \lambda - k_3 \end{vmatrix} = 0.$$

   Simplifying:

   $$(\lambda - (k_1 + k_2))(\lambda - k_3) - (-k_3)(-k_2) = 0,$$
   $$(\lambda - (k_1 + k_2))(\lambda - k_3) - k_3 k_2 = 0.$$

   Expanding:

   $$\lambda^2 - \lambda(k_1 + k_2 + k_3) + (k_1 + k_2)k_3 - k_3 k_2 = 0.$$
   $$\lambda^2 - \lambda(k_1 + k_2 + k_3) + k_1 k_3 = 0.$$

Step 3: Stability Conditions

Step 3: Stability Conditions

For stability, all eigenvalues must have negative real parts.

1. Condition for k_1:

$$k_1 < 0,$$

   since k_1 is an eigenvalue.

2. Conditions for the roots of the quadratic equation: The roots of the quadratic equation depend on the discriminant:

$$D = (k_1 + k_2 + k_3)^2 - 4(k_1 k_3).$$

   For the eigenvalues to have negative real parts:

   - The trace of the matrix B, which is the sum of eigenvalues, must be negative:

$$k_1 + k_2 + k_3 < 0.$$

   - The determinant of the matrix B, which is the product of eigenvalues, must be positive:

$$k_1 k_3 > 0.$$

Final Requirements on k_1, k_2, k_3:

1. $k_1 < 0$,
2. $k_3 < 0$,
3. $k_1 + k_2 + k_3 < 0$.

These conditions ensure that all eigenvalues of the system matrix A have negative real parts, guaranteeing stability.

Summary:

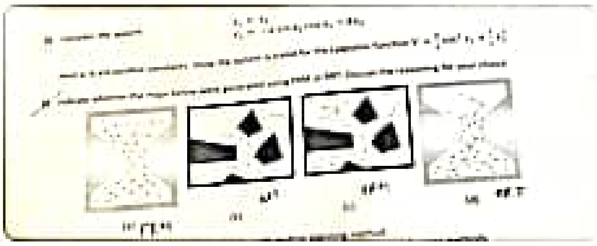
The system will be stable if the constants k_1, k_2, k_3 satisfy:

1. $k_1 < 0$,

Summary:

The system will be stable if the constants k_1, k_2, k_3 satisfy:

1. $k_1 < 0$,
2. $k_3 < 0$,
3. $k_1 + k_2 + k_3 < 0$.



Answer the question 1 & 2 with full description and explain elaborately everything.(PrinciplesofRobotAutonomy)

Question 1: Stability Analysis Using Lyapunov Function

Given System:

$$\dot{x}_1 = x_2,$$

$$\dot{x}_2 = -a\sin(x_1)\cos(x_1) - bx_2,$$

where $a > 0$ and $b > 0$.

Lyapunov Function:

The Lyapunov candidate function is given as:

$$V = \frac{a}{2}\sin^2(x_1) + \frac{1}{2}x_2^2.$$

Step 1: Verify Positivity of V

The Lyapunov function V must be positive definite. For this function:

1. The term $\frac{a}{2}\sin^2(x_1) \geq 0$, and it equals only when $\sin(x_1) = 0$, i.e., $x_1 = n\pi$, where n is an integer.

Question 1: *Stability Analysis Using Lyapunov Function*

Given System:

$$\dot{x}_1 = x_2,$$

$$\dot{x}_2 = -a\sin(x_1)\cos(x_1) - bx_2,$$

where $a > 0$ and $b > 0$.

Lyapunov Function:

The Lyapunov candidate function is given as:

$$V = \frac{a}{2}\sin^2(x_1) + \frac{1}{2}x_2^2.$$

Step 1: Verify Positivity of V

The Lyapunov function V must be positive definite. For this function:

1. The term $\frac{a}{2}\sin^2(x_1) \geq 0$, and it equals 0 only when $\sin(x_1) = 0$, i.e., $x_1 = n\pi$, where n is an integer.
2. The term $\frac{1}{2}x_2^2 \geq 0$, and it equals 0 only when $x_2 = 0$.

Thus:

$$V \geq 0 \quad \text{and} \quad V = 0 \iff (x_1, x_2) = (n\pi, 0).$$

Step 2: Compute the Derivative of V

Taking the time derivative of V along the trajectories of the system:

$$\dot{V} = \frac{\partial V}{\partial x_1}\dot{x}_1 + \frac{\partial V}{\partial x_2}\dot{x}_2.$$

For $V = \frac{a}{2}\sin^2(x_1) + \frac{1}{2}x_2^2$:

- $\frac{\partial V}{\partial x_1} = a\sin(x_1)\cos(x_1)$,
- $\frac{\partial V}{\partial x_2} = x_2$.

$$\dot{V} = \frac{\partial V}{\partial x_1}\dot{x}_1 + \frac{\partial V}{\partial x_2}\dot{x}_2.$$

For $V = \frac{a}{2}\sin^2(x_1) + \frac{1}{2}x_2^2$:

- $\frac{\partial V}{\partial x_1} = a\sin(x_1)\cos(x_1)$,
- $\frac{\partial V}{\partial x_2} = x_2$.

Substituting:

$$\dot{V} = (a\sin(x_1)\cos(x_1))\,\dot{x}_1 + x_2\dot{x}_2.$$

Using the system equations $\dot{x}_1 = x_2$ and $\dot{x}_2 = -a\sin(x_1)\cos(x_1) - bx_2$, substitute:

\dot{V} = \left(a \sin(x_1) \cos(x_1)\right)x_2 + x_2\left(-a \sin(x_1) \cos(x_1) - b x_2\).

Simplify:

$$\dot{V} = a\sin(x_1)\cos(x_1)x_2 - a\sin(x_1)\cos(x_1)x_2 - bx_2^2.$$

$$\dot{V} = -bx_2^2.$$

Step 3: Analyze $\dot{V}$

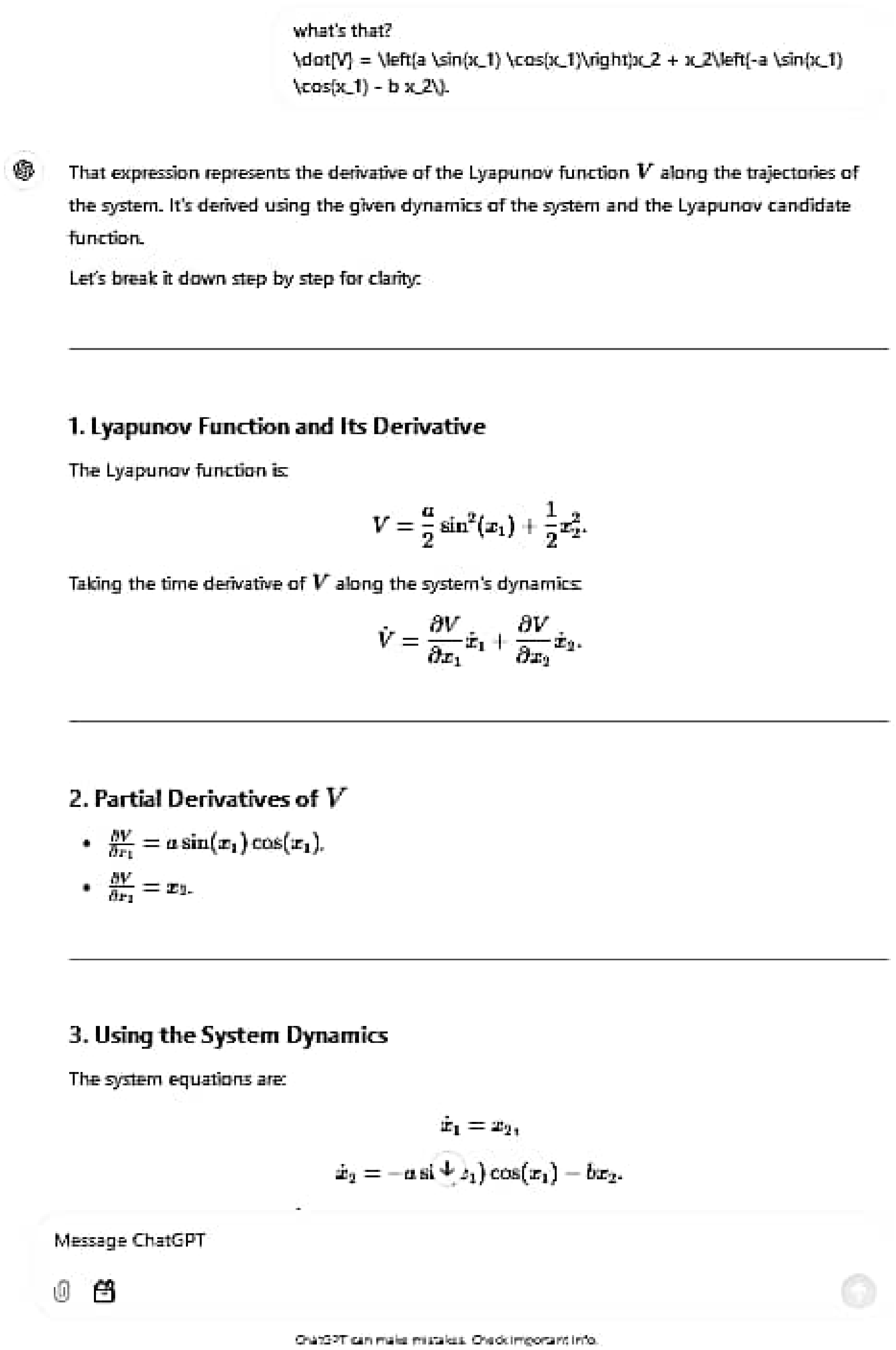
1. The term $-bx_2^2 \leq 0$, and it equals 0 only when $x_2 = 0$.
2. Since $\dot{V} \leq 0$, V is non-increasing. Additionally, $\dot{V} = 0$ only at $x_2 = 0$.

Step 4: Conclusion

Using Lyapunov's direct method, the system is stable because:

1. V is positive definite.
2. $\dot{V}$ is negative semi-definite.

Further, since the derivative $\dot{V} = 0$ only at isolated points ($x_1 = n\pi, x_2 = 0$), the system is asymptotically stable.



what's that?
\dot{V} = \left(a \sin(x_1) \cos(x_1)\right)x_2 + x_2\left(-a \sin(x_1) \cos(x_1) - b x_2\).

That expression represents the derivative of the Lyapunov function V along the trajectories of the system. It's derived using the given dynamics of the system and the Lyapunov candidate function.

Let's break it down step by step for clarity:

---

1. Lyapunov Function and Its Derivative

The Lyapunov function is:

$$V = \frac{a}{2}\sin^2(x_1) + \frac{1}{2}x_2^2.$$

Taking the time derivative of V along the system's dynamics:

$$\dot{V} = \frac{\partial V}{\partial x_1}\dot{x}_1 + \frac{\partial V}{\partial x_2}\dot{x}_2.$$

---

2. Partial Derivatives of V

- $\frac{\partial V}{\partial x_1} = a\sin(x_1)\cos(x_1)$.
- $\frac{\partial V}{\partial x_2} = x_2$.

---

3. Using the System Dynamics

The system equations are:

$$\dot{x}_1 = x_2,$$
$$\dot{x}_2 = -a\sin(x_1)\cos(x_1) - bx_2.$$

Message ChatGPT

ChatGPT can make mistakes. Check important info.

---

3. Using the System Dynamics

The system equations are:

$$\dot{x}_1 = x_2,$$

$$\dot{x}_2 = -a\sin(x_1)\cos(x_1) - bx_2.$$

Substitute $\dot{x}_1$ and $\dot{x}_2$ into $\dot{V}$:

$$\dot{V} = \left(a\sin(x_1)\cos(x_1)\right)\dot{x}_1 + x_2\dot{x}_2.$$

Substitute $\dot{x}_1 = x_2$ and $\dot{x}_2 = -a\sin(x_1)\cos(x_1) - bx_2$:

\dot{V} = \left(a \sin(x_1) \cos(x_1)\right)x_2 + x_2\left(-a \sin(x_1) \cos(x_1) - b x_2\).

---

4. Simplify the Expression

Expand the terms:

$$\dot{V} = a\sin(x_1)\cos(x_1)x_2 - a\sin(x_1)\cos(x_1)x_2 - bx_2^2.$$

Notice that $a\sin(x_1)\cos(x_1)x_2$ cancels out because it appears with opposite signs:

$$\dot{V} = -bx_2^2.$$

Final Result

The derivative of the Lyapunov function is:

$$\dot{V} = -bx_2^2.$$

This is negative semi-definite ($\dot{V} \leq 0$), confirming that the Lyapunov function decreases or remains constant over time, which is a key step in proving stability.

Question 2: PRM vs RRT Maps

Explanation of PRM and RRT:

1. Probabilistic Roadmap (PRM):
   - PRM constructs a roadmap in free space by sampling random points and connecting them if the path between them is collision-free.
   - PRM is well-suited for multi-query problems because the roadmap can be reused for different start and goal configurations.
2. Rapidly-Exploring Random Tree (RRT):
   - RRT incrementally builds a tree from the start configuration by sampling random points and connecting them to the nearest existing tree node.
   - RRT is better for single-query problems, especially in high-dimensional spaces.

Identify Each Map:

(a) PRM:

- The roadmap is fully constructed, connecting several nodes even in regions where paths might not be used immediately.
- The structure shows a dense graph, characteristic of PRM.

(b) RRT:

- The tree-like structure expands outward from a starting point and does not densely cover the space.
- This is typical of RRT, which focuses on expanding towards the goal rather than covering all of the free space.

(c) PRM:

- A roadmap with multiple connections, even in unused regions, again characteristic of PRM.

(d) RRT:

- Similar to (b), the tree expands outward, focusing on reaching a specific target, which aligns with RRT behavior.

- The structure shows a dense graph, characteristic of PRM.

(b) RRT:

- The tree-like structure expands outward from a starting point and does not densely cover the space.
- This is typical of RRT, which focuses on expanding towards the goal rather than covering all of the free space.

(c) PRM:

- A roadmap with multiple connections, even in unused regions, again characteristic of PRM.

(d) RRT:

- Similar to (b), the tree expands outward, focusing on reaching a specific target, which aligns with RRT behavior.

Summary:

1. Map (a): PRM
2. Map (b): RRT
3. Map (c): PRM
4. Map (d): RRT

Reasoning:

- PRM creates dense graphs, suitable for multi-query scenarios.
- RRT focuses on tree-like structures that prioritize single-query pathfinding.

what's that?
\dot{V} = \left(a \sin(x_1) \cos(x_1)\right)x_2 + x_2\left(-a \sin(x_1) \cos(x_1) - b x_2\).

That expression represents the derivative of the Lyapunov function V along the trajectories of the system. It's derived using the given dynamics of the system and the Lyapunov candidate

Edge Detection Algorithm: Overview

Edge detection is a process used to identify the boundaries or edges of objects within an image. The edges represent significant transitions in intensity (e.g., brightness) and often correspond to the boundaries of objects. The Canny Edge Detection Algorithm is a popular and efficient technique for this task.

Steps of Canny Edge Detection Algorithm

1. Convert the Image to Grayscale

- Many edge detection algorithms work on grayscale images since color information is not necessary for detecting intensity gradients.
- Mathematically, convert a color image $I(x, y)$ to grayscale:

  $$I_{gray}(x, y) = 0.299R(x, y) + 0.587G(x, y) + 0.114B(x, y)$$

  where R, G, and B are the red, green, and blue channels, respectively.

2. Apply Gaussian Smoothing

- To reduce noise, apply a Gaussian filter to smooth the image:

  $$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

  - σ: Standard deviation of the Gaussian distribution.
  - Convolve I_{gray} with the Gaussian kernel $G(x, y)$:

    $$I_{smooth}(x, y) = I_{gray}(x, y) * G(x, y)$$

3. Compute Gradients Using Sobel Filters

- Calculate the gradient of the image intensity to find edges. Use Sobel operators to approximate partial derivatives:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad , \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

  - Convolve I_{smooth} with G_x and G_y to compute the gradients:

$$I_x = I_{smooth} * G_x, \quad I_y = I_{smooth} * G_y$$

- Compute the magnitude and direction of the gradient:

$$|G| = \sqrt{I_x^2 + I_y^2}, \quad \theta = \tan^{-1}\left(\frac{I_y}{I_x}\right)$$

---

4. Non-Maximum Suppression

- Thin the edges to one-pixel width by suppressing non-maximum values in the gradient direction.
- For each pixel, check the two neighboring pixels in the gradient direction (θ) and suppress the pixel if it is not greater than its neighbors.

---

5. Double Thresholding

- Use two thresholds: T_{low} and T_{high} to classify edges as:
  - Strong edges: $|G| > T_{high}$
  - Weak edges: $T_{low} < |G| \leq T_{high}$

5. Double Thresholding

- Use two thresholds: T_{low} and T_{high} to classify edges as:
    - Strong edges: $|G| > T_{high}$
    - Weak edges: $T_{low} < |G| \leq T_{high}$
    - Non-edges: $|G| \leq T_{low}$

---

6. Edge Tracking by Hysteresis

- Connect weak edges to strong edges if they are adjacent, ensuring that weak edges are only preserved if they are part of a significant edge structure.

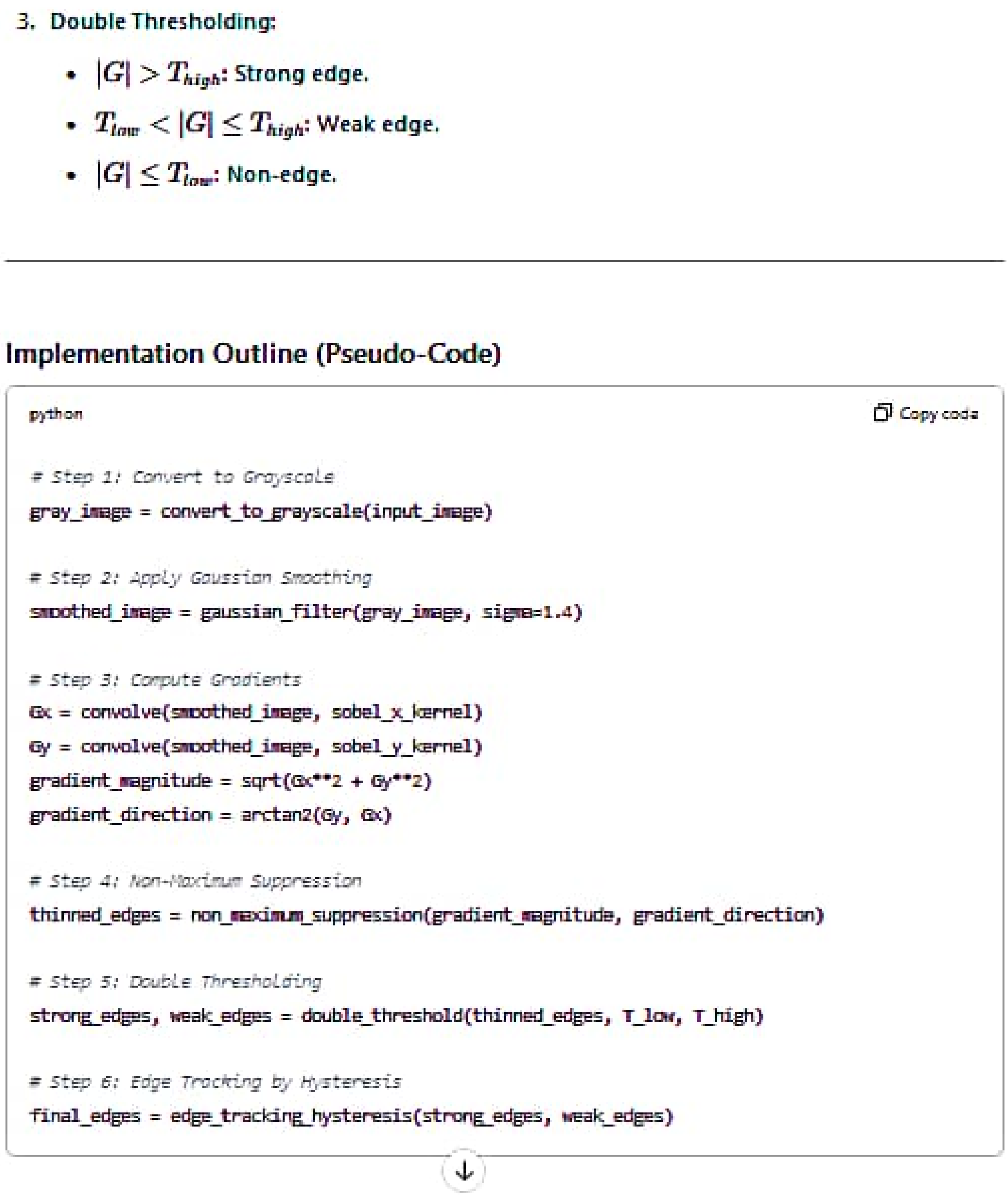
Mathematical Summary

1. Gradient Computation:

$$G_x = I_{smooth} * G_x, \quad G_y = I_{smooth} * G_y$$

$$|G| = \sqrt{I_x^2 + I_y^2}, \quad \theta = \tan^{-1}\left(\frac{I_y}{I_x}\right)$$

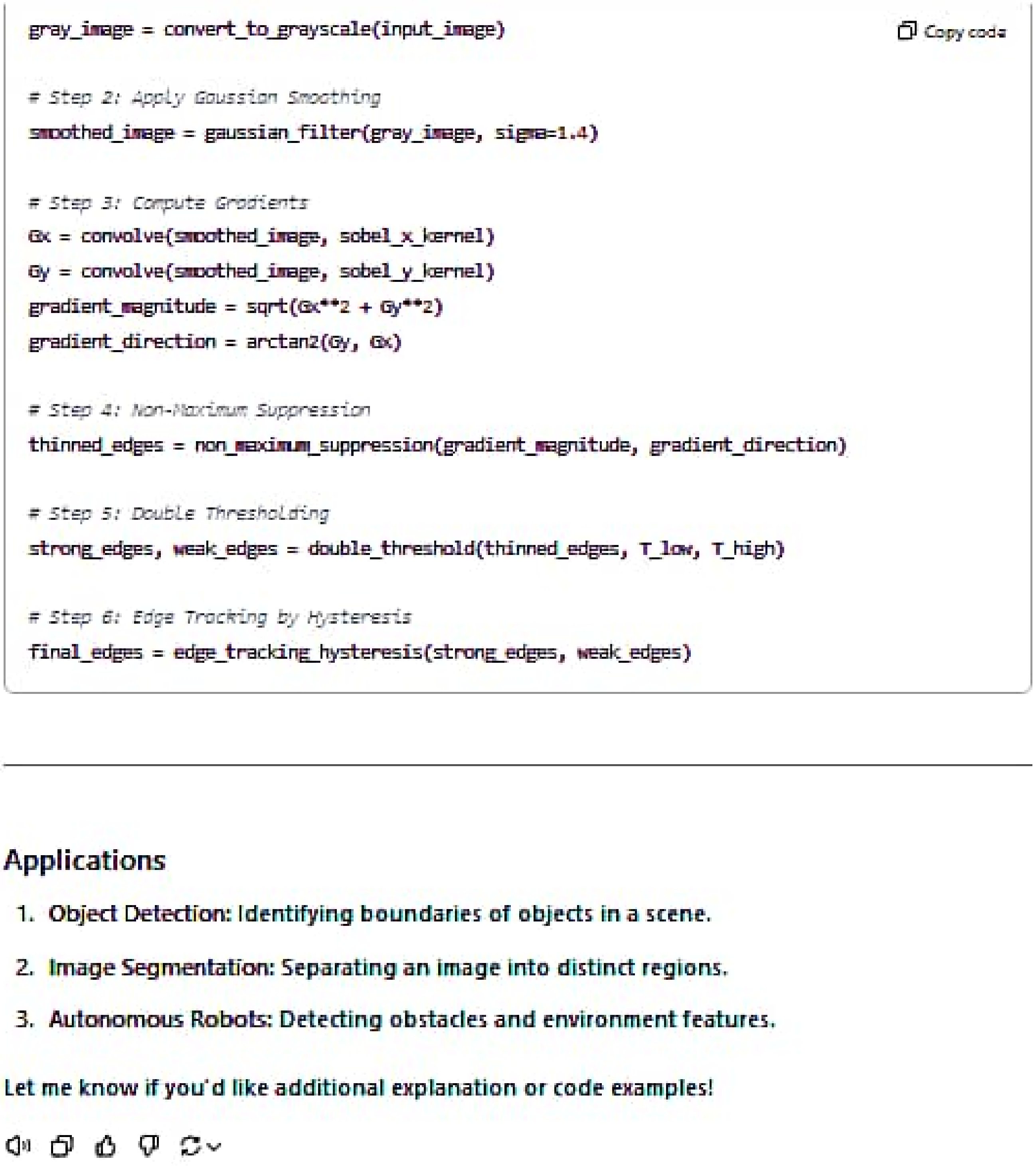
2. Non-Maximum Suppression: Compare each pixel to its neighbors along θ.
3. Double Thresholding:
    - $|G| > T_{high}$: Strong edge.
    - $T_{low} < |G| \leq T_{high}$: Weak edge.
    - $|G| \leq T_{low}$: Non-edge.



3. **Double Thresholding:**
    - $|G| > T_{high}$: Strong edge.
    - $T_{low} < |G| \leq T_{high}$: Weak edge.
    - $|G| \leq T_{low}$: Non-edge.

Implementation Outline (Pseudo-Code)

```python
# Step 1: Convert to Grayscale
gray_image = convert_to_grayscale(input_image)

# Step 2: Apply Gaussian Smoothing
smoothed_image = gaussian_filter(gray_image, sigma=1.4)

# Step 3: Compute Gradients
Gx = convolve(smoothed_image, sobel_x_kernel)
Gy = convolve(smoothed_image, sobel_y_kernel)
gradient_magnitude = sqrt(Gx**2 + Gy**2)
gradient_direction = arctan2(Gy, Gx)

# Step 4: Non-Maximum Suppression
thinned_edges = non_maximum_suppression(gradient_magnitude, gradient_direction)

# Step 5: Double Thresholding
strong_edges, weak_edges = double_threshold(thinned_edges, T_low, T_high)

# Step 6: Edge Tracking by Hysteresis
final_edges = edge_tracking_hysteresis(strong_edges, weak_edges)
```



```
gray_image = convert_to_grayscale(input_image)

# Step 2: Apply Gaussian Smoothing
smoothed_image = gaussian_filter(gray_image, sigma=1.4)

# Step 3: Compute Gradients
Gx = convolve(smoothed_image, sobel_x_kernel)
Gy = convolve(smoothed_image, sobel_y_kernel)
gradient_magnitude = sqrt(Gx**2 + Gy**2)
gradient_direction = arctan2(Gy, Gx)

# Step 4: Non-Maximum Suppression
thinned_edges = non_maximum_suppression(gradient_magnitude, gradient_direction)

# Step 5: Double Thresholding
strong_edges, weak_edges = double_threshold(thinned_edges, T_low, T_high)

# Step 6: Edge Tracking by Hysteresis
final_edges = edge_tracking_hysteresis(strong_edges, weak_edges)
```

Applications

1. **Object Detection**: Identifying boundaries of objects in a scene.
2. **Image Segmentation**: Separating an image into distinct regions.
3. **Autonomous Robots**: Detecting obstacles and environment features.

Let me know if you'd like additional explanation or code examples!

Question 3: Given an input image, describe an algorithm to implement an edge detector. If possible, use mathematical equations to describe the steps.

Overview of Edge Detection

Edge detection is the process of identifying points in an image where the intensity changes sharply. These points often correspond to object boundaries, surface discontinuities, or changes in texture.

The purpose of edge detection in robotics autonomy is to help identify obstacles, map environments, or detect objects in dynamic environments.

Steps of Edge Detection Using the Sobel Operator

Step 1: Convert the Image to Grayscale

The input image, usually in RGB format, is converted to grayscale because color information is unnecessary for edge detection.

Mathematical Conversion:

$$I_{\text{gray}}(x, y) = 0.299 \cdot R(x, y) + 0.587 \cdot G(x, y) + 0.114 \cdot B(x, y)$$

Step 2: Apply Gaussian Smoothing

To reduce noise that can create false edges, apply a Gaussian filter to smooth the image. This reduces high-frequency components while preserving the overall structure.

Gaussian Smoothing Kernel:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Here, σ controls the level of smoothing.

Step 3: Compute the Gradient Magnitude and Direction

The Sobel operator is used to approximate the derivatives of the intensity function. This detects regions of rapid intensity change (edges) by calculating gradients in the x- and y-directions.

Gradient in x-direction (Horizontal Edge

Gradient in x-direction (Horizontal Edges):

$$G_x = I_{gray} * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Gradient in y-direction (Vertical Edges):

$$G_y = I_{gray} * \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Gradient Magnitude and Direction:

$$|G| = \sqrt{G_x^2 + G_y^2}, \quad \theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Step 4: Non-Maximum Suppression

Suppress all gradient values that are not local maxima in the gradient direction (θ). This step reduces thick edges to one-pixel wide lines.

1. Quantize the gradient directions (θ) to one of four angles: 0°, 45°, 90°, or 135°.
2. For each pixel, compare its gradient magnitude to its neighbors along the gradient direction. Suppress it if it is not greater than both neighbors.

Step 5: Apply Double Threshold

To distinguish significant edges from noise, use two thresholds:

- T_{high}: High threshold to detect strong edges.
- T_{low}: Low threshold to detect weak edges that are connected to strong edges.

Classify pixels as:

- Strong Edge Pixels: Gradient magnitude $|G| > T_{high}$.
- Weak Edge Pixels: $T_{low} < |G| \leq T_{high}$.
- Non-Edge Pixels: Gradient magnitude $|G| \leq T_{low}$.

Step 6: Edge Tracking by Hysteresis

Weak edges that are connected to strong edges are preserved, while isolated weak edges are suppressed.

1. Traverse the weak edges.
2. Retain those connected to a strong edge pixel.

1. Traverse the weak edges.
2. Retain those connected to a strong edge pixel.

Mathematical Representation

1. Smoothing:

$$I_{smooth} = I_{gray} * G(x, y)$$

2. Gradient computation:

$$G_x = I_{smooth} * G_x, \quad G_y = I_{smooth} * G_y$$

3. Gradient magnitude and direction:

$$|G| = \sqrt{G_x^2 + G_y^2}, \quad \theta = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

4. Non-maximum suppression: Suppress gradient values where:

$$|G(i, j)| < \max(|G(i+1, j)|, |G(i-1, j)|)$$

   along the gradient direction.

5. Double thresholding: Assign pixels as strong, weak, or non-edges based on T_{high} and T_{low}.

Canny Edge Detection Implementation (Popular Algorithm)

Here is an outline of the Canny Edge Detection algorithm, which integrates the steps described above:

1. Smooth the image with a Gaussian filter.
2. Compute gradients using Sobel operators.
3. Apply non-maximum suppression.
4. Use double thresholding to identify strong and weak edges.
5. Perform edge tracking by hysteresis.

Python Code Example

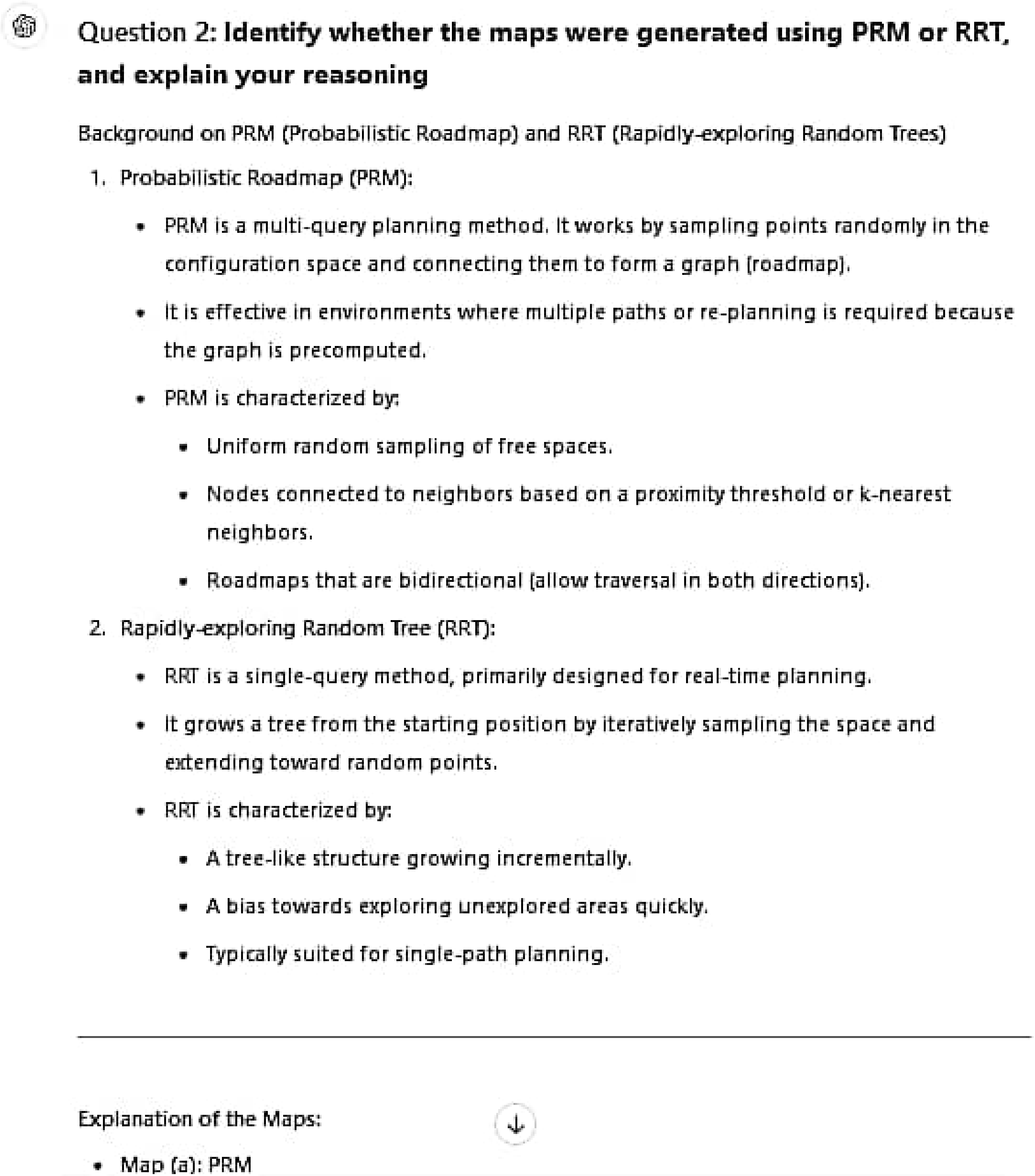
python Copy code

```python
import cv2
import numpy as np

def canny_edge_detector(image, low_threshold, high_threshold):
    # Step 1: Convert to Grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

    # Step 2: Apply Gaussian Blur
    blurred = cv2.GaussianBlur(gray, (5, 5), 1.4)

    # Step 3: Compute Gradients
    Gx = cv2.Sobel(blurred, cv2.CV_64F, 1, 0, ksize=3)
    Gy = cv2.Sobel(blurred, cv2.CV_64F, 0, 1, ksize=3)
    gradient_magnitude = np.sqrt(Gx**2 + Gy**2)
    gradient_direction = np.arctan2(Gy, Gx)

    # Step 4: Non-Maximum Suppression
    edges = cv2.Canny(gray, low_threshold, high_threshold)

    return edges

# Example usage
image = cv2.imread("input_image.jpg")
edges = canny_edge_detector(image, 50, 150)
cv2.imshow("Edges", edges)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Applications

1. **Obstacle Detection**: Detecting sharp transitions in intensity to identify obstacles.
2. **Object Recognition**: Defining object boundaries for feature extraction.
3. **Mapping in Robotics**: Identifying edges to create accurate environment maps.

Let me know if further clarification or additional examples are required!



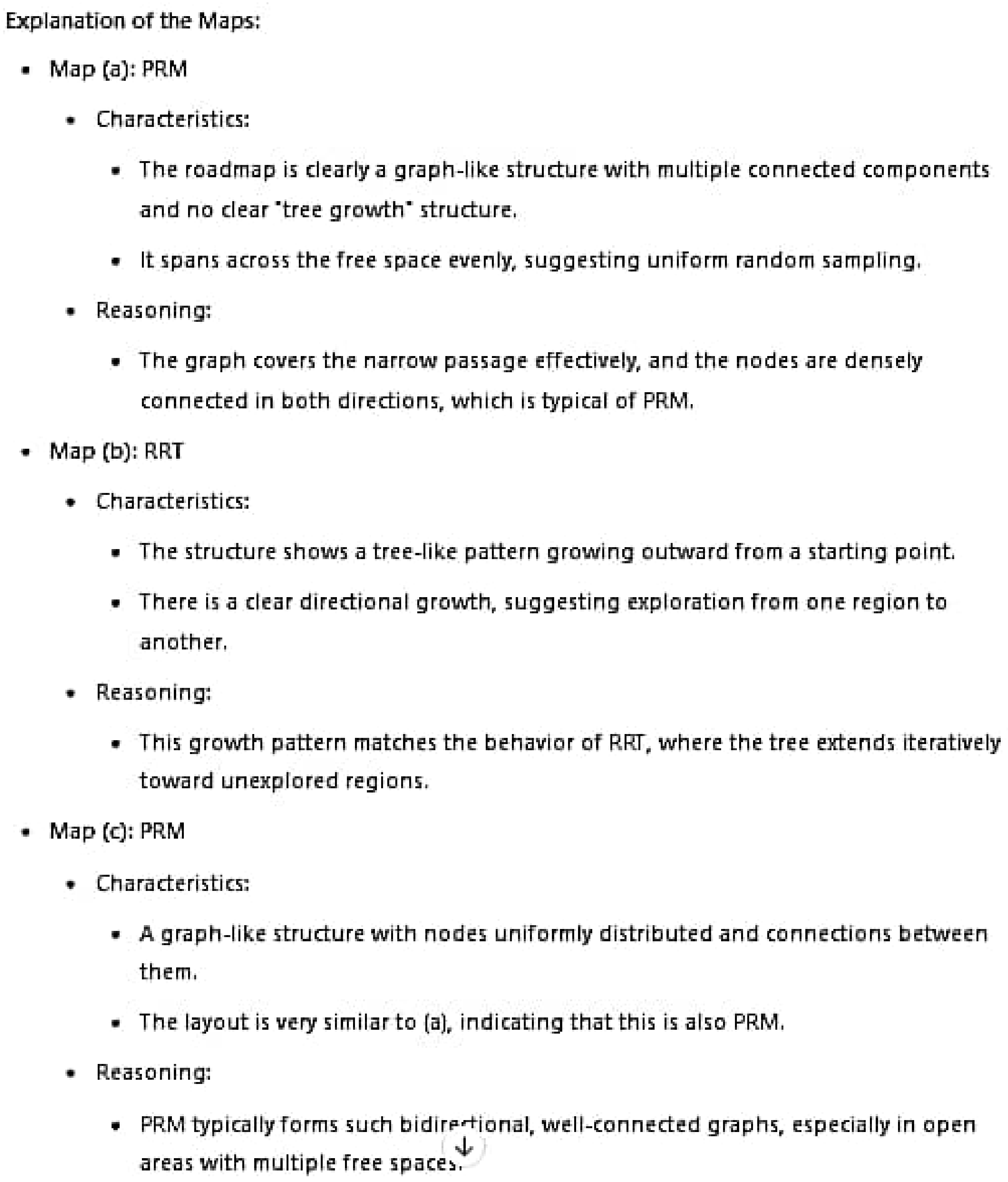
Question 2: **Identify whether the maps were generated using PRM or RRT, and explain your reasoning**

Background on PRM (Probabilistic Roadmap) and RRT (Rapidly-exploring Random Trees)

1. Probabilistic Roadmap (PRM):
    - PRM is a multi-query planning method. It works by sampling points randomly in the configuration space and connecting them to form a graph (roadmap).
    - It is effective in environments where multiple paths or re-planning is required because the graph is precomputed.
    - PRM is characterized by:
        - Uniform random sampling of free spaces.
        - Nodes connected to neighbors based on a proximity threshold or k-nearest neighbors.
        - Roadmaps that are bidirectional (allow traversal in both directions).
2. Rapidly-exploring Random Tree (RRT):
    - RRT is a single-query method, primarily designed for real-time planning.
    - It grows a tree from the starting position by iteratively sampling the space and extending toward random points.
    - RRT is characterized by:
        - A tree-like structure growing incrementally.
        - A bias towards exploring unexplored areas quickly.
        - Typically suited for single-path planning.

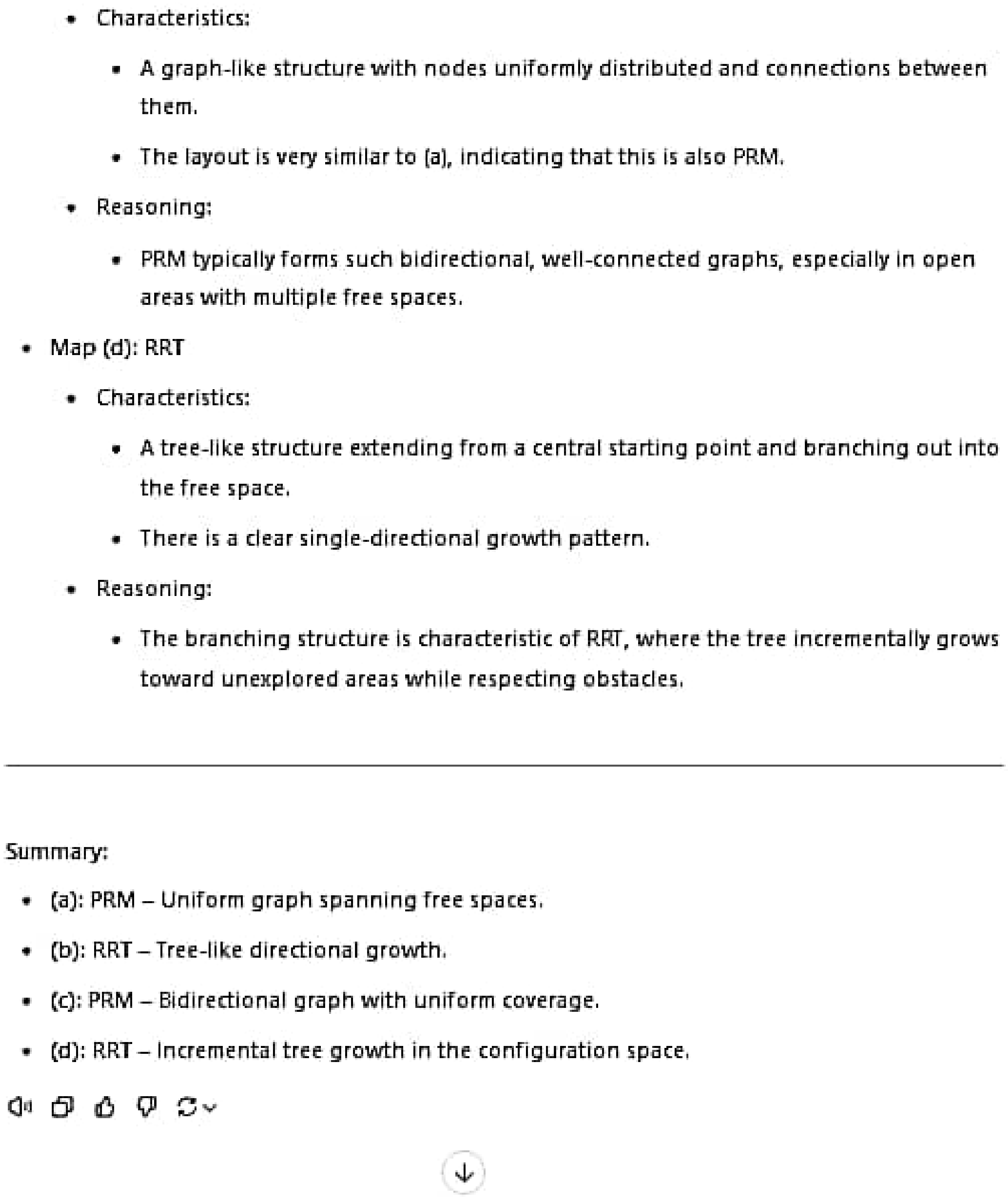
Explanation of the Maps:

- Map (a): PRM



Explanation of the Maps:

- Map (a): PRM
  - Characteristics:
    - The roadmap is clearly a graph-like structure with multiple connected components and no clear "tree growth" structure.
    - It spans across the free space evenly, suggesting uniform random sampling.
  - Reasoning:
    - The graph covers the narrow passage effectively, and the nodes are densely connected in both directions, which is typical of PRM.
- Map (b): RRT
  - Characteristics:
    - The structure shows a tree-like pattern growing outward from a starting point.
    - There is a clear directional growth, suggesting exploration from one region to another.
  - Reasoning:
    - This growth pattern matches the behavior of RRT, where the tree extends iteratively toward unexplored regions.
- Map (c): PRM
  - Characteristics:
    - A graph-like structure with nodes uniformly distributed and connections between them.
    - The layout is very similar to (a), indicating that this is also PRM.
  - Reasoning:
    - PRM typically forms such bidirectional, well-connected graphs, especially in open areas with multiple free spaces.



  - Characteristics:
    - A graph-like structure with nodes uniformly distributed and connections between them.
    - The layout is very similar to (a), indicating that this is also PRM.
  - Reasoning:
    - PRM typically forms such bidirectional, well-connected graphs, especially in open areas with multiple free spaces.
- Map (d): RRT
  - Characteristics:
    - A tree-like structure extending from a central starting point and branching out into the free space.
    - There is a clear single-directional growth pattern.
  - Reasoning:
    - The branching structure is characteristic of RRT, where the tree incrementally grows toward unexplored areas while respecting obstacles.

Summary:

- (a): PRM – Uniform graph spanning free spaces.
- (b): RRT – Tree-like directional growth.
- (c): PRM – Bidirectional graph with uniform coverage.
- (d): RRT – Incremental tree growth in the configuration space.

Step-by-Step Explanation of the A* Grid-Based Search Motion Planning Method

A* (A-star) is one of the most popular algorithms for motion planning in robotics. It is a complete and optimal search algorithm that uses both the cost to reach a node (g-cost) and a heuristic estimate of the cost to the goal (h-cost) to determine the best path. Here's a detailed breakdown of how it works in a grid-based environment:

---

1. Setup the Environment

- Grid Representation: Divide the space into a grid. Each cell represents a discrete state (free or occupied).
- Define Start and Goal Nodes:
  - Start Node: The initial position of the robot in the grid.
  - Goal Node: The target position to be reached.
- Assign Costs:
  - g: The cost to move from the start node to the current node.
  - h: The heuristic estimate of the cost to move from the current node to the goal (explained in Step 5).
  - $f = g + h$: The total estimated cost to move through the current node to the goal.

---

2. Initialize Data Structures

- Open List: A priority queue that holds the nodes to be evaluated (starts with the start

2. Initialize Data Structures

- Open List: A priority queue that holds all the nodes to be evaluated (starts with the start node).
- Closed List: A list of nodes that have already been evaluated.
- Initially:
    - Add the start node to the open list.
    - The closed list is empty.

---

3. Repeat Until the Goal is Found

While the open list is not empty:

- Step 3.1: Pick the Node with the Lowest f-Value.
    - Extract the node with the smallest $f = g + h$ from the open list. This is the current node.
    - If this node is the goal, stop. The path is found.
- Step 3.2: Move the Node to the Closed List.
    - Mark the current node as "visited" by adding it to the closed list. It will not be evaluated again.

---

4. Expand Neighbors

- For each neighboring cell of the current node:
    - Skip Obstacles: If the neighbor is occupied (an obstacle) or already in the closed list,

4. **Expand Neighbors**

- For each neighboring cell of the current node:
  - **Skip Obstacles:** If the neighbor is occupied (an obstacle) or already in the closed list, skip it.
  - **Calculate Costs for Free Neighbors:**
    - Compute g: The movement cost from the start node to this neighbor. This depends on the type of movement:
      - Straight movement (up, down, left, right) typically costs 1.
      - Diagonal movement typically costs $\sqrt{2}$.
    - Compute h: The heuristic estimate (explained in Step 5).
    - Compute $f = g + h$.

---

5. **Heuristic Function**

The heuristic guides the search by estimating the cost to reach the goal from a node. Common heuristics include:

- **Manhattan Distance:** $h = |x_{\text{goal}} - x_{\text{current}}| + |y_{\text{goal}} - y_{\text{current}}|$ (used for grid maps without diagonal movement).
- **Euclidean Distance:** $h = \sqrt{(x_{\text{goal}} - x_{\text{current}})^2 + (y_{\text{goal}} - y_{\text{current}})^2}$ (used for grid maps with diagonal movement).
- **Diagonal Distance:** $h = \max(|x_{\text{goal}} - x_{\text{current}}|, |y_{\text{goal}} - y_{\text{current}}|)$.

---

6. Update the Open List

- If the neighbor is not in the open list:
  - Add it to the open list and record its g, h, and f values.
  - Set the current node as its parent (to reconstruct the path later).
- If the neighbor is already in the open list but the new g-value is smaller:
  - Update its g-value, f-value, and parent node.

---

7. Path Reconstruction

- Once the goal node is reached, backtrack from the goal to the start node using the parent pointers recorded in Step 6.
- This reconstructed path is the optimal solution.

Example Walkthrough

Problem:

- A 5x5 grid with obstacles.
- Start at $(0, 0)$, goal at $(4, 4)$.
- Obstacles block cells $(2, 2)$ and $(3, 2)$.

Execution:

1. Start at $(0, 0)$, calculate $f = g + h$ for neighbors $(0, 1)$ and $(1, 0)$.
2. Move to the neighbor with the lowest [illegible]alue (tie-breaking based on h).
3. Add obstacles $(2, 2)$ and $(3, 2)$ to the closed list.

2. Move to the neighbor with the lowest f-value (tie-breaking based on h).
3. Add obstacles $(2,2)$ and $(3,2)$ to the closed list.
4. Expand nodes until $(4,4)$ is reached.
5. Backtrack to find the path: $(0,0) \rightarrow (1,0) \rightarrow (2,0) \rightarrow (3,1) \rightarrow (4,2) \rightarrow (4,4)$.

Advantages of A*:

1. **Optimality:** Guaranteed to find the shortest path (if the heuristic is admissible and consistent).
2. **Efficiency:** By using heuristics, it reduces the number of nodes explored compared to uninformed searches like Dijkstra.
3. **Flexibility:** Can be adapted with different heuristics for specific environments.

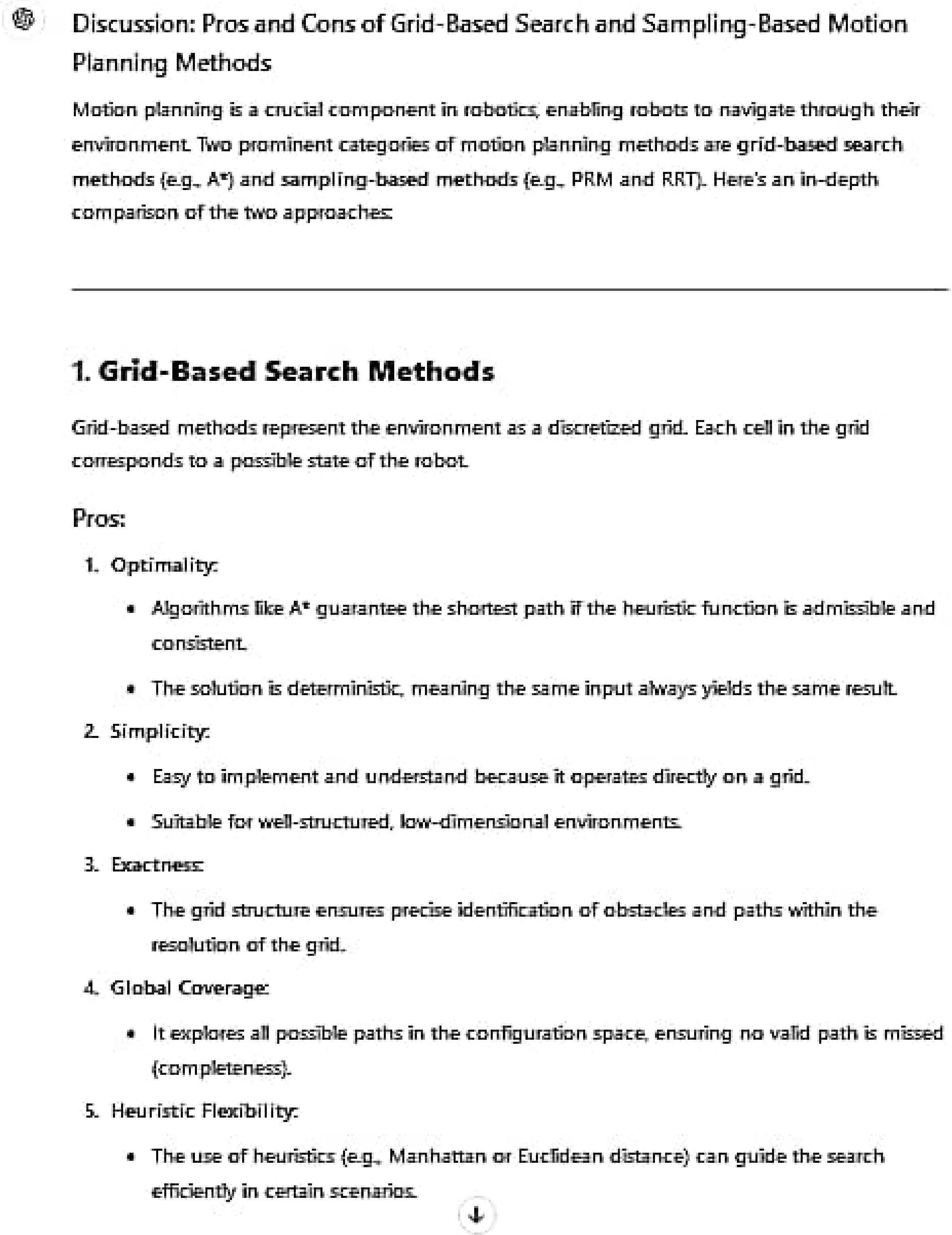
Disadvantages of A*:

1. **Memory Usage:** Requires storing all nodes in memory, making it unsuitable for very large grids.
2. **Performance:** Can be slower in high-dimensional spaces or with complex obstacles.

Applications:

- Robot pathfinding (e.g., autonomous vacuum cleaners or drones).
- Game AI for navigating characters.
- Real-world navigation in maps and GPS systems.

Let me know if you want me to illustrate this with an example grid!



Discussion: Pros and Cons of Grid-Based Search and Sampling-Based Motion Planning Methods

Motion planning is a crucial component in robotics, enabling robots to navigate through their environment. Two prominent categories of motion planning methods are grid-based search methods (e.g., A*) and sampling-based methods (e.g., PRM and RRT). Here's an in-depth comparison of the two approaches:

---

1. Grid-Based Search Methods

Grid-based methods represent the environment as a discretized grid. Each cell in the grid corresponds to a possible state of the robot.

Pros:

1. Optimality:
    - Algorithms like A* guarantee the shortest path if the heuristic function is admissible and consistent.
    - The solution is deterministic, meaning the same input always yields the same result.
2. Simplicity:
    - Easy to implement and understand because it operates directly on a grid.
    - Suitable for well-structured, low-dimensional environments.
3. Exactness:
    - The grid structure ensures precise identification of obstacles and paths within the resolution of the grid.
4. Global Coverage:
    - It explores all possible paths in the configuration space, ensuring no valid path is missed (completeness).
5. Heuristic Flexibility:
    - The use of heuristics (e.g., Manhattan or Euclidean distance) can guide the search efficiently in certain scenarios.

Cons:

1. Scalability:
    - The number of grid cells grows exponentially with the number of dimensions, leading to high computational and memory requirements (curse of dimensionality).
2. Resolution Dependency:
    - Paths are constrained by the grid resolution. Low resolution may overlook narrow passages, while high resolution increases computation.
3. High Computational Cost:
    - Exact exploration of all nodes can be slow, especially in large environments.
4. Poor Suitability for Complex Configuration Spaces:
    - For high-dimensional spaces (e.g., robotic arms), representing the environment as a grid becomes impractical.

---

2. Sampling-Based Motion Planning Methods

Sampling-based methods, like Probabilistic Roadmap (PRM) and Rapidly-exploring Random Tree (RRT), generate random samples in the configuration space and connect them to construct a roadmap or a tree.

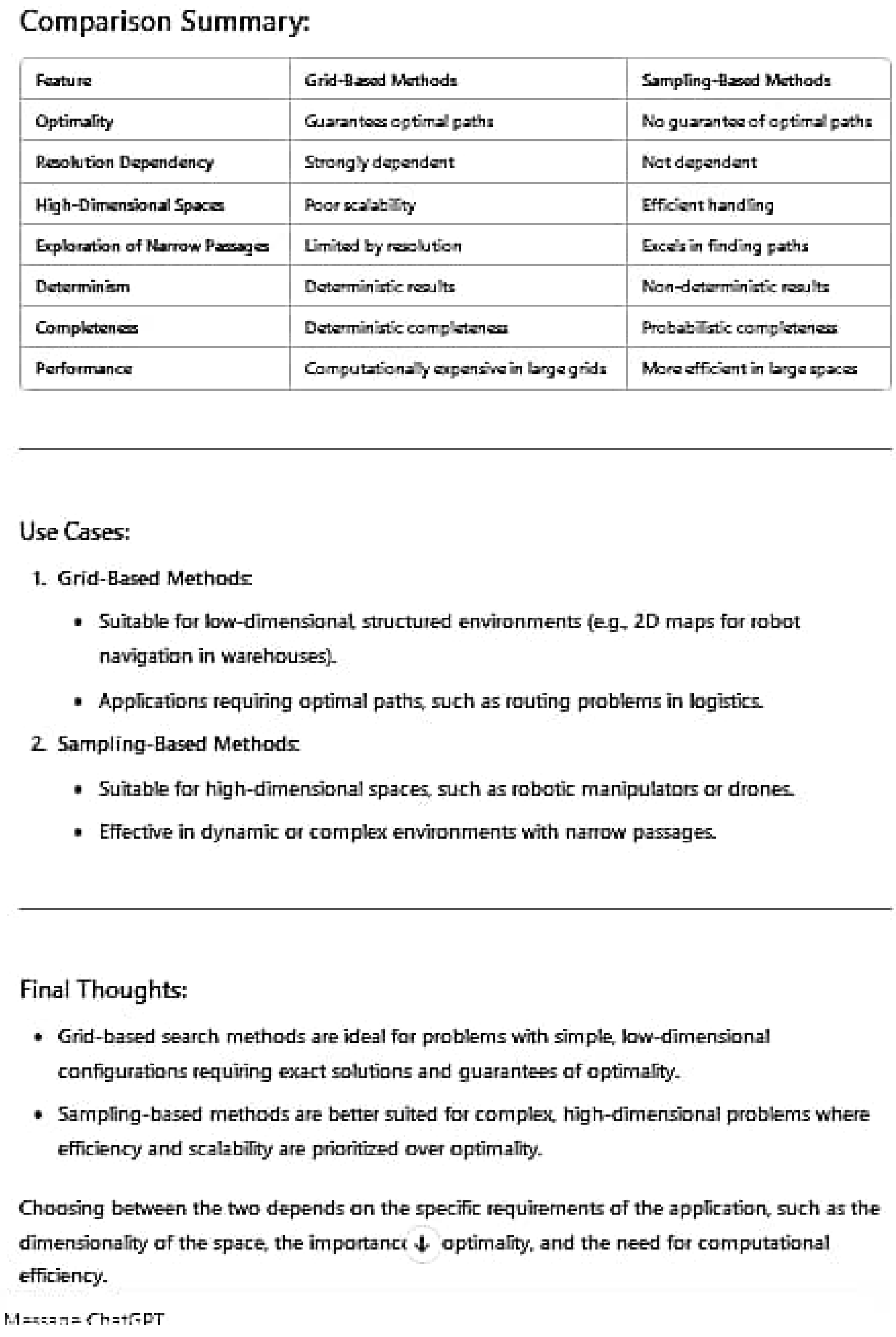
Pros:

1. Scalability to High Dimensions:
    - Sampling-based methods handle high-dimensional configuration spaces more effectively than grid-based approaches.
    - Suitable for robotic arms, drones, and manipulators.
2. Efficient Handling of Narrow Passages:
    - Algorithms like PRM and RRT excel at finding paths in environments with narrow corridors, where grid-based methods might struggle due to discretization.
3. Faster Exploration:
    - Sampling methods do not require a predefined resolution and can quickly explore large spaces without explicitly discretizing them.

3. **Faster Exploration:**
   - Sampling methods do not require a predefined resolution and can quickly explore large spaces without explicitly discretizing them.
4. **Probabilistic Completeness:**
   - Given enough time, they are likely to find a solution if one exists.
5. **Adaptability:**
   - Sampling-based methods can adapt to dynamic environments by re-sampling or updating the roadmap/tree.

Cons:

1. **Suboptimality:**
   - The solution path is often not the shortest or smoothest due to the randomness in sampling.
   - Post-processing (e.g., smoothing) is usually required to improve the path quality.
2. **Lack of Determinism:**
   - Results may vary across runs due to the randomness inherent in the algorithm.
3. **No Guarantees of Exactness:**
   - Unlike grid-based methods, sampling-based planners are not guaranteed to find the optimal path, even if one exists.
4. **Heavy Dependency on Sampling Density:**
   - Low sampling density can lead to missed narrow passages or obstacles.
   - Increasing the sampling density improves the solution quality but increases computational cost.
5. **Graph Connectivity Issues (PRM-specific):**
   - In PRM, sparsely sampled graphs may result in disconnected components, making some regions of the space unreachable.



Comparison Summary:

Feature	Grid-Based Methods	Sampling-Based Methods
Optimality	Guarantees optimal paths	No guarantee of optimal paths
Resolution Dependency	Strongly dependent	Not dependent
High-Dimensional Spaces	Poor scalability	Efficient handling
Exploration of Narrow Passages	Limited by resolution	Excels in finding paths
Determinism	Deterministic results	Non-deterministic results
Completeness	Deterministic completeness	Probabilistic completeness
Performance	Computationally expensive in large grids	More efficient in large spaces

---

Use Cases:

1. Grid-Based Methods:
    - Suitable for low-dimensional, structured environments (e.g., 2D maps for robot navigation in warehouses).
    - Applications requiring optimal paths, such as routing problems in logistics.
2. Sampling-Based Methods:
    - Suitable for high-dimensional spaces, such as robotic manipulators or drones.
    - Effective in dynamic or complex environments with narrow passages.

Final Thoughts:

- Grid-based search methods are ideal for problems with simple, low-dimensional configurations requiring exact solutions and guarantees of optimality.
- Sampling-based methods are better suited for complex, high-dimensional problems where efficiency and scalability are prioritized over optimality.

Choosing between the two depends on the specific requirements of the application, such as the dimensionality of the space, the importanc optimality, and the need for computational efficiency.