CSE5280 Assignment: Penalty Functions for Floor Plan Navigation

Eraldo Ribeiro

February 10, 2026

Department of Electrical Engineering and Computer Science
Florida Institute of Technology
Melbourne, FL, U.S.A.

Contents

1 Overview

In this assignment, you will extend the cost-function minimization framework described in the *Animation by Cost-Function Minimization* notebook to a two-dimensional floor plan consisting of walls rather than point obstacles.

- **Notebook Animation by Cost-Function Minimization**: https://github.com/eraldoribeiro/cse5280_animation_by_cost_function_minimization

Your goal is to model walls as continuous penalty fields, combine them with a goal-attraction term, and use gradient descent to generate collision-free motion through the environment. In this assignment, walls are not geometric constraints—they are features of the cost landscape. Your animation succeeds or fails based entirely on how well you design that landscape.

This assignment emphasizes *problem formulation*: the animation should emerge entirely from how you define and combine cost functions.

1.1 Scope Clarification

This assignment is **not**:

- A shortest-path planning problem
- A graph-search problem
- A discrete collision-detection task

You are **not expected** to compute globally optimal paths or completeness guarantees.

Instead, the focus is on:

- Continuous cost-field design
- Local, gradient-based motion
- Understanding how motion emerges from cost functions

1.2 Learning Objectives

By completing this assignment, you will be able to:

- Represent walls as continuous penalty functions
- Convert geometric constraints into scalar cost fields
- Compute and visualize gradients of spatial cost functions
- Generate motion using gradient descent in complex environments
- Analyze the strengths and limitations of potential-field methods

1.3 Problem Description

You are given a simple two-dimensional floor plan composed of:

- Axis-aligned walls (line segments)
- A start position
- A goal position

Your task is to design penalty functions that:

- Penalize proximity to walls
- Prevent the animated point from crossing through walls
- Remain differentiable so gradient-based optimization can be applied

The final motion should reach the goal while remaining collision-free.

Expected Behavior

A successful implementation should demonstrate:

- No visible wall crossings.
- Convergence toward the goal for reasonable parameter choices.
- Qualitatively different paths when wall parameters are changed.

1.4 Deliverables

You must submit a link to a **GitHub repository** containing your work.

The repository should contain at minimum:

- `notebook.ipynb`, which includes:
  - Wall penalty function implementations
  - Gradient descent implementation
  - Visualizations and animations
  - A comparison of **at least three** different wall penalty formulations, using the same floor plan and start/goal configuration. The comparison should include both visual results (e.g., trajectories, cost fields) and a qualitative discussion of behavior.
- Generated figures, including:
  - Cost contours
  - Gradient vector field
  - Trajectory overlaid on the floor plan

Your notebook should include sufficient **Markdown explanations** to clearly describe your approach, design choices, and analysis.

1.5 Constraints and Guidelines

- Do not use discrete collision detection or path planning algorithms (e.g., A*, RRT).
- Motion must be produced *only* by cost-function minimization.
- You may not hardcode collision checks (e.g., "if collision then stop").
- All avoidance behavior must emerge from your penalty functions.

1.6 Evaluation Criteria

Criterion	**Weight**
Correct modeling of walls as penalty fields	30%
Quality of cost-field visualizations	20%
Correct use of gradients	20%
Analysis and explanation	20%
Code clarity and organization	10%

2 Task Breakdown

2.1 Floor Plan Representation

- Represent each wall as a line segment defined by two endpoints.
- Store the full floor plan as a collection of wall segments.

You may assume:

- Walls do not move.
- Walls have a finite thickness or influence radius.

Throughout this assignment, "wall thickness" and "influence radius" refer to a user-defined distance parameter controlling how far a wall's penalty extends.

2.2 Wall Penalty Function

For each wall, define a penalty function $C_{\text{wall}}(\mathbf{x})$ that depends on the distance from a point $\mathbf{x}$ to the wall.

Your penalty function must:

- Increase as the point approaches the wall
- Be zero beyond a chosen influence distance
- Be smooth enough to compute gradients numerically

You may adapt or extend penalty functions used in the notebook (e.g., logarithmic or inverse-distance penalties).

Gradients may be computed analytically or numerically (e.g., finite differences). You are not required to derive closed-form gradients for all penalty terms. You may also use library functions for gradient computation if desired.

2.3 Combined Cost Function

Define the total cost as

$$C(\mathbf{x}) = C_{\text{goal}}(\mathbf{x}) + \sum_i w_i \, C_{\text{wall},i}(\mathbf{x}), \tag{2.1}$$

where:

- $C_{\text{goal}}(\mathbf{x})$ is an attraction term pulling the point toward the goal,
- $C_{\text{wall},i}(\mathbf{x})$ are wall penalty terms,
- w_i are weighting coefficients.

Parameter tuning (e.g., weights, influence radius) is an expected part of this assignment and should be discussed in the report.

2.4 Visualization of Cost Fields

Produce the following visualizations:

- A contour plot of the total cost function over the floor plan
- A vector field showing the negative gradient of the total cost
- A plot showing walls, start, and goal positions

These visualizations should clearly show how walls shape the cost landscape.

2.5 Motion Generation via Gradient Descent

- Implement gradient descent to animate a point starting from the given start position.
- Use the negative gradient of the total cost to update the position iteratively.
- Stop the animation when the point reaches the goal or after a fixed number of iterations.

It is acceptable—and expected—for some configurations to produce local minima. Part of this assignment is understanding why these arise.

2.6 Analysis

Answer the following questions:

1. How does wall thickness or influence radius affect the resulting path?
2. What happens near corners or narrow corridors?
3. Does the method ever get stuck? Why or why not?

3 Penalty Functions to Implement and Compare

In this assignment, you will implement and compare **multiple wall penalty functions**. All penalty functions must be based on the same geometric primitive: **Distance from a Point to a Wall Segment**.

Walls are defined as line segments:

$$s_i = [\mathbf{a}_i, \mathbf{b}_i], \tag{3.1}$$

and the distance from a point $\mathbf{x} \in \mathbb{R}^2$ to a wall segment is:

$$d_i(\mathbf{x}) = \min_{t \in [0,1]} \|\mathbf{x} - (\mathbf{a}_i + t(\mathbf{b}_i - \mathbf{a}_i))\| \, . \tag{3.2}$$

Details on how to calculated Equation 3.2 are described in Section 3.1.

All wall costs follow the same structure:

$$C_{\text{walls}}(\mathbf{x}) = \sum_i w_i \, \phi(d_i(\mathbf{x})) \, , \tag{3.3}$$

where $\phi(d)$ is a scalar penalty function.

Your task is to **experiment with different choices of $\phi(d)$** and analyze how they change the resulting motion. You must implement and compare **at least three** of the penalty functions listed below.

3.1 How to calculate the distance function

The computation of :

$$d_i(\mathbf{x}) = \min_{t \in [0,1]} \|\mathbf{x} - (\mathbf{a}_i + t(\mathbf{b}_i - \mathbf{a}_i))\| \tag{3.4}$$

can be done as follows:

1. Project the point onto the infinite line through the segment
2. Restrict the projection to stay within the segment endpoints

3. Measure the Euclidean distance to that closest point

Mathematically, these steps are described as follows.

Distance from a Point to a Line Segment

Let $\mathbf{x} \in \mathbb{R}^2$ be a point, and let a wall segment be defined by its endpoints

$$s = [\mathbf{a}, \mathbf{b}], \qquad \mathbf{a}, \mathbf{b} \in \mathbb{R}^2. \tag{3.5}$$

Step 1: Segment parameterization Any point on the segment can be written as

$$\mathbf{p}(t) = \mathbf{a} + t(\mathbf{b} - \mathbf{a}), \qquad t \in [0, 1]. \tag{3.6}$$

Step 2: Distance to a point on the segment The squared distance from $\mathbf{x}$ to a point on the segment is

$$f(t) = \left\|\mathbf{x} - (\mathbf{a} + t(\mathbf{b} - \mathbf{a}))\right\|^2. \tag{3.7}$$

Minimizing $f(t)$ is equivalent to minimizing the Euclidean distance.

Step 3: Unconstrained projection Define the vectors

$$\mathbf{v} = \mathbf{b} - \mathbf{a}, \qquad \mathbf{w} = \mathbf{x} - \mathbf{a}. \tag{3.8}$$

The minimizer of $f(t)$ over the infinite line is

$$t^\star = \frac{\mathbf{w} \cdot \mathbf{v}}{\mathbf{v} \cdot \mathbf{v}} \tag{3.9}$$

Step 4: Clamp to the segment Since the segment is finite, the valid parameter is

$$t = \operatorname{clamp}(t^\star, 0, 1), \tag{3.10}$$

where

$$\operatorname{clamp}(z, 0, 1) = \min\Big(1, \max(0, z)\Big). \tag{3.11}$$

Step 5: Closest point on the segment The closest point on the segment is

$$\mathbf{q} = \mathbf{a} + t\,\mathbf{v}. \tag{3.12}$$

Step 6: Distance to the segment The distance from $\mathbf{x}$ to the segment is

$$d(\mathbf{x}, s) = \|\mathbf{x} - \mathbf{q}\|. \tag{3.13}$$

Final expression Combining the steps above, the distance from a point to a line segment is

$$d(\mathbf{x}, s) = \left\| \mathbf{x} - \left(\mathbf{a} + \operatorname{clamp}\left(\frac{(\mathbf{x} - \mathbf{a}) \cdot (\mathbf{b} - \mathbf{a})}{\|\mathbf{b} - \mathbf{a}\|^2}, 0, 1 \right) (\mathbf{b} - \mathbf{a}) \right) \right\|. \tag{3.14}$$

Degenerate case If $\mathbf{a} = \mathbf{b}$, the segment reduces to a single point and the distance is simply

$$d(\mathbf{x}, s) = \|\mathbf{x} - \mathbf{a}\|. \tag{3.15}$$

The non-vectorized version of the point-to-segment distance computation is shown in Listing 3.1. A version of this code is implemented in the Colab notebook provided with this assignment.

```python
import numpy as np

def point_segment_distance(x, a, b):
    """
    Compute the minimum Euclidean distance from a point x to
    the line segment defined by endpoints a and b.

    This implements:
        d(x, [a,b]) = min_{t in [0,1]} || x - (a + t(b-a)) ||

    Parameters
    ----------
    x : array_like, shape (2,)
        Query point.
    a : array_like, shape (2,)
        Start point of the segment.
    b : array_like, shape (2,)
        End point of the segment.
    Returns
    -------
    d : float
        Distance from x to the segment [a,b].
    """

    # Convert inputs to NumPy arrays (ensures vector math works)
    x = np.asarray(x, dtype=float)
    a = np.asarray(a, dtype=float)
    b = np.asarray(b, dtype=float)

    # Vector along the segment from a to b
    v = b - a

    # Vector from a to the query point x
    w = x - a

    # Squared length of the segment
    vv = np.dot(v, v)

    # Handle degenerate case: segment reduces to a single point
    if vv == 0.0:
        return np.linalg.norm(x - a)

    # Project w onto v to find the optimal parameter t
    t = np.dot(w, v) / vv

    # Clamp t to the segment interval [0, 1]
    t = np.clip(t, 0.0, 1.0)

    # Compute the closest point on the segment
    q = a + t * v

    # Distance from x to the closest point q
    d = np.linalg.norm(x - q)

    return d
```

Listing 3.1: Python implementation of the point-to-segment distance used for wall penalty functions.

3.2 Candidate Wall Penalty Functions

Below are several approved penalty function families. All are valid; none is considered "the correct one." Your analysis should focus on how their *qualitative behavior* differs.

3.3 Penalty Function Families to Try

3.3.1 Truncated Log Barrier (Strong Repulsion)

$$\phi(d) = \begin{cases} \log\left(\dfrac{R}{d+\varepsilon}\right), & d \le R, \\ 0, & d > R. \end{cases} \tag{3.16}$$

Behavior

- Very strong repulsion near the wall
- Effectively creates a "forbidden" region close to walls

Discussion Points

- Sensitivity to ε
- Behavior near corners
- Numerical stiffness

3.3.2 Truncated Inverse-Distance Repulsion

$$\phi(d) = \begin{cases} \dfrac{1}{(d+\varepsilon)^p}, & d \le R, \\ 0, & d > R, \end{cases} \qquad p \in \{1, 2\}. \tag{3.17}$$

Behavior

- Smooth but increasingly aggressive near the wall
- Adjustable sharpness via p

Discussion Points

- Comparison with the log barrier
- Gradient magnitude near walls
- Stability versus safety

3.3.3 Quadratic Band (Soft Wall)

$$\phi(d) = \begin{cases} \frac{1}{2}(R-d)^2, & d \le R, \\ 0, & d > R. \end{cases} \tag{3.18}$$

Behavior

- Smooth gradients
- Creates a "buffer zone" around walls

Discussion Points

- Why trajectories may "graze" walls
- Trade-off between smoothness and safety

3.3.4 Quartic Band (Sharper Soft Wall)

$$\phi(d) = \begin{cases} (R-d)^4, & d \le R, \\ 0, & d > R. \end{cases} \tag{3.19}$$

Behavior

- Stronger repulsion than the quadratic band
- Fully smooth and bounded

Discussion Points

- How higher-order penalties change path curvature
- Comparison with inverse-distance penalties

3.3.5 Exponential / Gaussian Repulsion

$$\phi(d) = \exp\left(-\frac{d^2}{2\sigma^2}\right), \tag{3.20}$$

optionally truncated for $d > R$.

Behavior

- Extremely smooth everywhere
- Long-range but weak influence

Discussion Points

- Why this formulation may require larger weights
- Why gradients never truly vanish without truncation

3.3.6 Signed-Distance (Capsule) Penetration Penalty

Walls have finite thickness.

Define the signed distance

$$\mathrm{sdf}_i(\mathbf{x}) = d_i(\mathbf{x}) - t, \tag{3.21}$$

where t is the wall half-thickness.

The penalty is defined as

$$\phi(\mathrm{sdf}) = [\max(0, -\mathrm{sdf})]^2 . \tag{3.22}$$

Behavior

- No penalty unless the wall is penetrated
- Clean physical interpretation

Discussion Points

- Difference between “repulsion” and “constraint violation”

- Comparison with buffer-zone penalties

3.3.7 Corridor-Friendly Penalty (Parallel Walls)

For two parallel walls forming a corridor, define the cost as

$$C(\mathbf{x}) = C_{\text{goal}}(\mathbf{x}) + w \sum_i \phi(d_i(\mathbf{x})) + \lambda \left(\text{sdf}_1(\mathbf{x}) - \text{sdf}_2(\mathbf{x})\right)^2 . \tag{3.23}$$

Behavior

- Prevents oscillation between walls
- Encourages motion along the corridor center

Discussion Points

- Why naive wall repulsion fails in narrow passages
- How adding structure changes local minima

3.4 Comparison Criteria

You are encouraged to compare penalty functions along the following axes:

- Path smoothness
- Clearance from walls
- Convergence speed
- Sensitivity to parameters
- Presence of local minima
- Numerical stability

3.5 Recommended Starting Defaults

For robust behavior, consider the following workflow:

- Start with the **Quadratic Band** penalty.
- If walls are crossed, try the **Log Barrier**.

- If motion becomes too stiff, try the **Quartic** or **Gaussian** penalties.

Typical parameter values (in map units):

- Influence radius R: 5–20% of the map width
- ε: 10^{-3}–10^{-2}
- Weight w: 10–100 relative to the goal cost scale

4 Floor Plan Examples

In this assignment, you will implement and compare different wall penalty functions using this floor plan. The following variations of the floor plan are provided to test specific behaviors:

- **Baseline**: The original L-shaped floor plan with a single corner.
- **Zigzag**: A narrow zig-zag corridor that tests the ability to navigate tight spaces without oscillation.
- **Dead-end**: A dead-end corridor that tests the ability to escape local minima.
- **Symmetric**: A symmetric obstacle configuration that tests the ability to break symmetry and avoid getting stuck.

Figure 4.1 shows a simple floor plan with walls, start, and goal positions. This L-shaped environment is the baseline for testing your penalty functions.

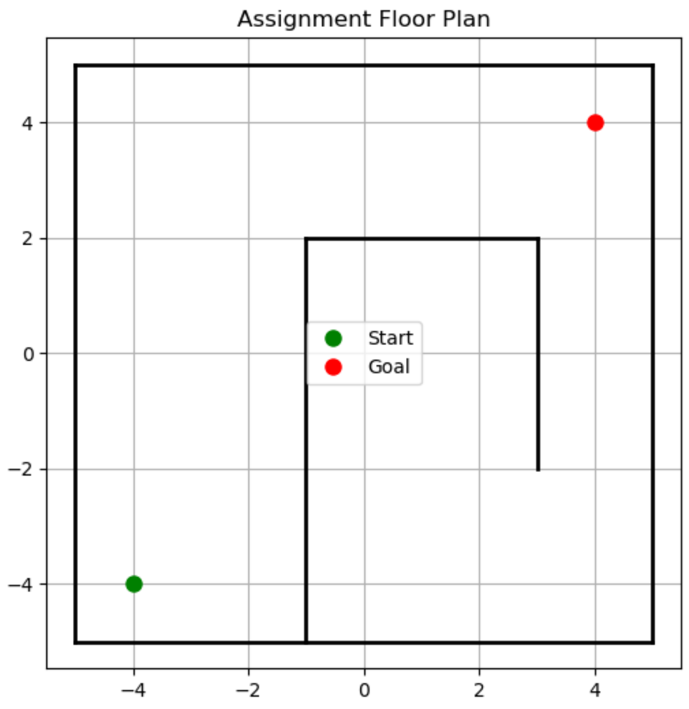

Figure 4.1: Baseline floor plan with walls (black), start (green), and goal (red).

Figure 4.2 shows the same floor plan with a sample cost field and gradient vector field overlaid. The cost contours illustrate how the walls shape the cost landscape, while the arrows indicate the direction of steepest descent. This visualization helps to understand how different penalty functions influence the motion generated by gradient descent.

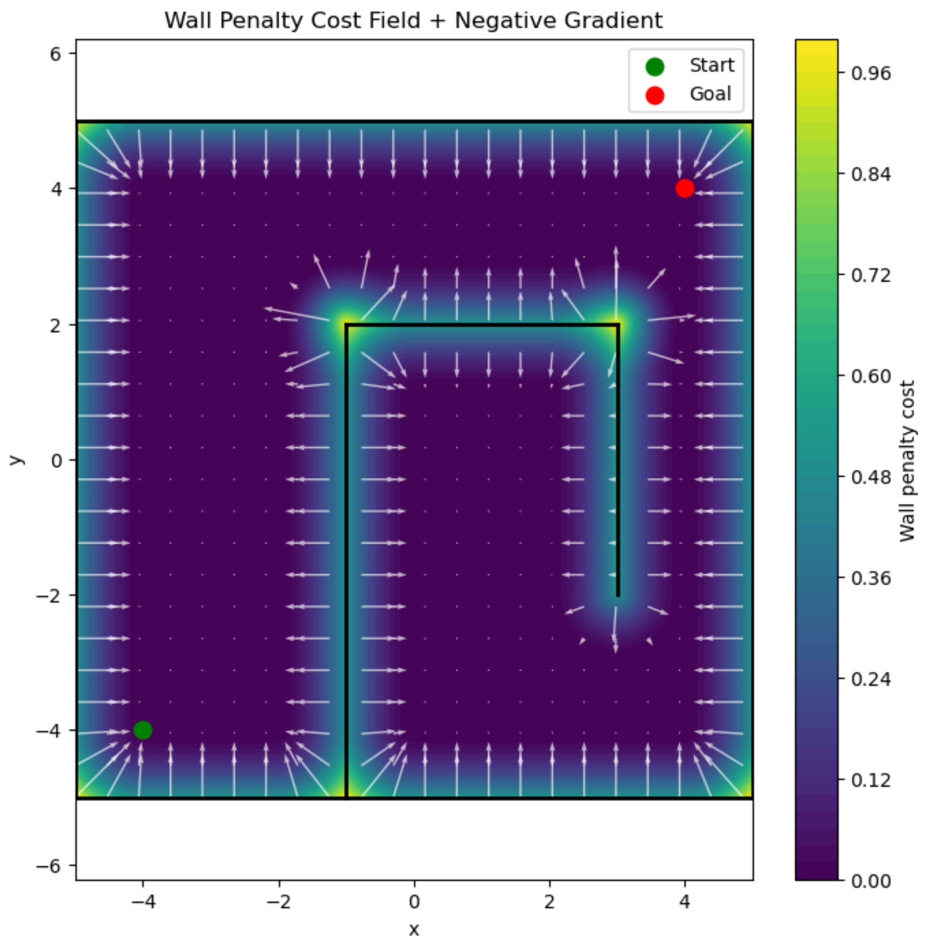

Figure 4.2: Cost contours and gradient vector field overlaid on the floor plan.

Figure 4.3 shows the resulting trajectory of the animated point as it moves from the start to the goal position under the influence of the combined cost function. The path should navigate around walls while converging toward the goal, demonstrating how the choice of penalty functions affects the motion.

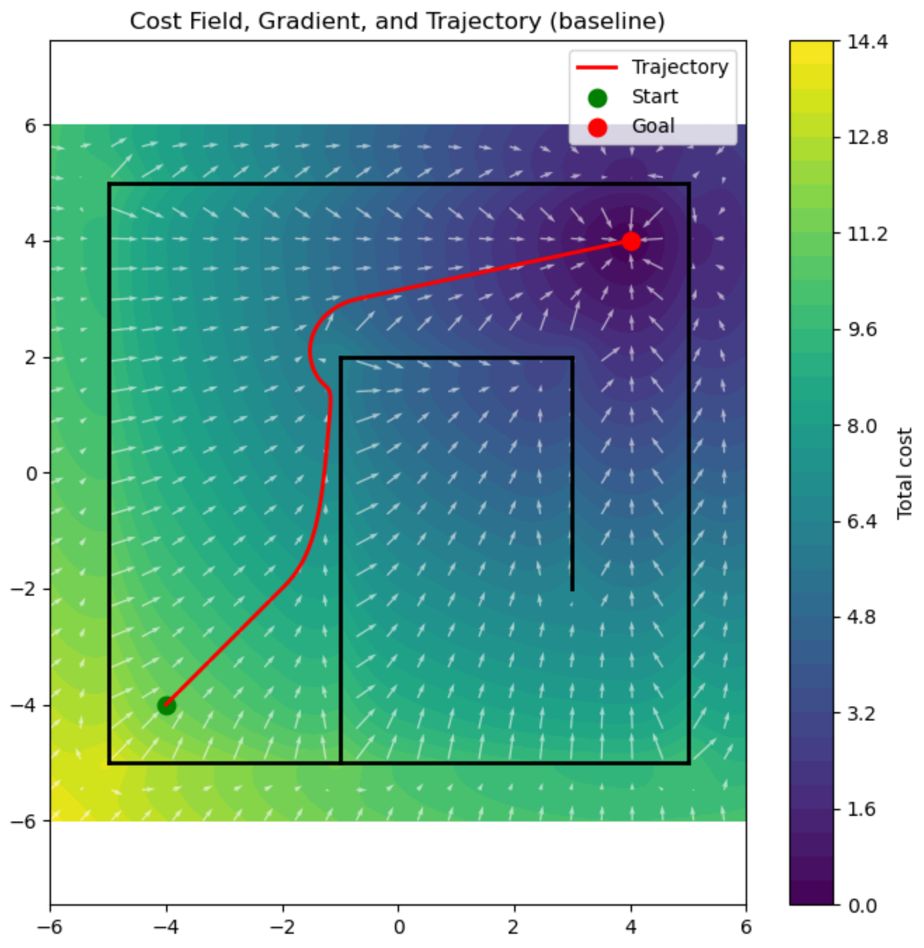

Figure 4.3: Trajectory generated by gradient descent on the combined cost function.