Eikonal PDE-Based Safe Path Planning: Fast Marching Implementation and Analysis

Hyeonjae Park
Georgia Institute of Technology

April 2025

1 Introduction

Safe path planning is the task of finding a route from a start location to a goal while meeting two objectives at the same time: travel efficiency, such as short distance or low energy, and hazard avoidance, such as obstacles or danger zones. In practice, for mobile robots, self driving vehicles, and industrial machines, the environment contains regions with many levels of risk. A path that follows the geometric shortest line can pass through areas that are too dangerous, whereas a path that avoids all danger can become unnecessarily long when time matters. The main challenge is to merge efficiency and safety into one coherent strategy.

This project adopts a continuous field formulation. A cost value is assigned to every point of a grid: high cost marks dangerous or forbidden areas, and low cost marks safe, open space. Starting at the source, the cost field expands outward like water flowing over uneven terrain, where the terrain height represents the difficulty of reaching each point. The expansion process satisfies the Eikonal equation, a partial differential equation that describes how a wave propagates through media with different speeds, which in this context correspond to varying levels of risk.

Once the cost field is known, a path is recovered by tracing backward from the goal to the start along the route of minimal accumulated cost. This backtrace automatically balances distance and danger. To compute the field efficiently, we use the fast marching method, which processes cells in a strict priority order and therefore avoids redundant updates, keeping the algorithm accurate and fast even on dense grids.

As an example, consider a mobile robot on a two dimensional grid. Cells that contain obstacles or hazards receive high cost, while cells in which the robot can move freely receive low cost. After the cost wave has expanded from the start, every cell stores the exact effort required to reach it, taking into account both distance and local risk. By following the steepest descent of this cost from the goal, the robot obtains a route that stays safe without making unnecessary detours.

2 Mathematical Problem Formulation

2.1 Domain Setup and Notation

Let $\Omega \subset \mathbb{R}^2$ denote the environment in which we seek a safe path. We assume Ω contains:

- A *start point* $\mathbf{s} \in \Omega$ and a *goal point* $\mathbf{g} \in \Omega$.
- A set of *obstacles*, denoted by $\mathcal{O} \subset \Omega$.
- A continuous *risk (or hazard) field*, which quantifies the local cost of being near or within obstacles.

We will formulate an *energy* (or *cost*) functional that incorporates both distance traveled and exposure to risky areas. By minimizing this cost, we obtain a path that balances efficiency and safety.

2.2 Cost Function $c(\mathbf{x})$

We define a scalar cost function $c : \Omega \to [0, \infty)$ such that larger values of $c(\mathbf{x})$ indicate higher risk or greater difficulty. A typical definition is:

$$c(\mathbf{x}) \;=\; \begin{cases} C_{\text{obs}}, & \mathbf{x} \in \mathcal{O}, \\ 1 + \alpha \exp\bigl(-\beta \operatorname{dist}(\mathbf{x}, \mathcal{O})\bigr), & \mathbf{x} \in \Omega \setminus \mathcal{O}, \end{cases} \tag{1}$$

where $\operatorname{dist}(\mathbf{x}, \mathcal{O})$ is the distance from $\mathbf{x}$ to the nearest point on the obstacle set $\mathcal{O}$, while $\alpha, \beta > 0$ are user-selected parameters. The constant C_{obs} is taken to be very large (or effectively infinite) for points inside obstacles, ensuring such regions become practically impassable.

2.3 Path Integral Representation of the Cost

We represent any candidate path from $\mathbf{s}$ to $\mathbf{g}$ by a continuous curve

$$\Gamma : [0, 1] \;\to\; \Omega, \qquad \Gamma(0) = \mathbf{s}, \quad \Gamma(1) = \mathbf{g}. \tag{2}$$

To measure how "safe" or "costly" the path Γ is, we consider the following line integral:

$$\mathcal{E}(\Gamma) \;=\; \int_0^1 c\bigl(\Gamma(t)\bigr)\, \bigl\|\Gamma'(t)\bigr\|\, \mathrm{d}t. \tag{3}$$

Here, $\|\Gamma'(t)\|\mathrm{d}t$ is an infinitesimal arc length element, while $c(\Gamma(t))$ scales that length by the local cost. Thus, traversing obstacle-adjacent zones, where $c(\mathbf{x})$ is large, yields a higher value of $\mathcal{E}(\Gamma)$, discouraging the path from passing near obstacles unless strictly necessary.

2.4 Variational Minimization Problem

The *safe path planning* task can be formalized as the following minimization:

$$\Gamma^* \;=\; \arg\min_{\Gamma} \bigl\{\; \mathcal{E}(\Gamma) \;:\; \Gamma(0) = \mathbf{s},\; \Gamma(1) = \mathbf{g} \bigr\}. \tag{4}$$

The curve Γ^* that achieves this minimum provides a globally optimal trade-off between path length and hazard avoidance.

2.5 Value Function Interpretation

An equivalent perspective arises by defining a *value function*

$$T : \Omega \;\to\; \mathbb{R}, \tag{5}$$

where $T(\mathbf{x})$ represents the minimal cost to travel from the start $\mathbf{s}$ to any point $\mathbf{x} \in \Omega$ under the cost metric (1). By construction, $T(\mathbf{s}) = 0$ and the quantity $T(\mathbf{g})$ corresponds exactly to the minimal path energy in (4). In a continuous sense,

$$T(\mathbf{x}) \;=\; \min_{\Gamma:\, \Gamma(0)=\mathbf{s},\, \Gamma(1)=\mathbf{x}} \int_0^1 c\big(\Gamma(t)\big)\, \|\Gamma'(t)\|\, \mathrm{d}t.$$

Solving for T over Ω can be more convenient in a PDE framework, since T itself must satisfy appropriate boundary conditions (e.g., $T(\mathbf{s}) = 0$) and a local consistency condition linking T's gradient to $c(\mathbf{x})$.

Thus, we have now posed the safe path planning problem in two complementary but equivalent ways:

- Minimizing the path energy (3) directly in (4).
- Finding a minimal-cost value function $T(\mathbf{x})$ that encodes $\mathcal{E}(\Gamma)$ implicitly.

These formulations set the stage for deriving a PDE-based numerical method that efficiently computes T (and hence Γ^*). In the following sections, we will derive such a PDE from this energy perspective, and then show how to discretize and implement it.

3 Deriving the Eikonal PDE from the Variational Formulation

In this section, we provide a very detailed, step-by-step derivation of the PDE directly from the energy (or cost) model introduced in Section 2. We will show how minimizing the path-based functional

$$\mathcal{E}(\Gamma) \;=\; \int_0^1 c\big(\Gamma(t)\big)\,\|\Gamma'(t)\|\,\mathrm{d}t \tag{6}$$

naturally yields the well-known Eikonal equation, which can then be solved numerically to obtain the global minimum-cost path between the start and goal.

3.1 Revisiting the Path Integral Formulation

We represent a path $\Gamma : [0,1] \to \Omega$ with fixed boundary conditions,

$$\Gamma(0) = \mathbf{s}, \quad \Gamma(1) = \mathbf{g}, \tag{7}$$

where $\mathbf{s}$ is the start and $\mathbf{g}$ is the goal location. The cost function $c(\mathbf{x}) \geq 0$ reflects the local "difficulty" or "risk." Our objective is:

$$\Gamma^* \;=\; \arg\min_{\Gamma} \left\{ \int_0^1 c\big(\Gamma(t)\big)\,\|\Gamma'(t)\|\,\mathrm{d}t \right\}. \tag{8}$$

The fundamental question is: *what PDE or differential condition emerges from this minimization?* To see it, we proceed in two ways:

1. **Direct Variation of the Curve Γ:** we treat Γ as the primary unknown, vary it, and identify the stationarity (Euler-Lagrange) condition. This is more traditional for parametric curves.
2. **Value Function $T(\mathbf{x})$:** we rewrite the problem in terms of $T(\mathbf{x})$, the minimal cost to reach $\mathbf{x}$ from $\mathbf{s}$, then show that $\|\nabla T(\mathbf{x})\| = c(\mathbf{x})$ must hold. This is the Eikonal PDE.

In practice, the second viewpoint leads more directly to a solvable PDE, but we outline both perspectives in detail to show precisely how the PDE arises.

3.2 Approach I: Classical Euler-Lagrange Derivation

3.2.1 Lagrangian and Variation Setup

Consider the integrand in (6):

$$\mathcal{L}\big(\Gamma(t), \Gamma'(t)\big) \;=\; c\big(\Gamma(t)\big)\,\|\Gamma'(t)\|. \tag{9}$$

For a small variation $\eta(t)$ of $\Gamma(t)$ (with $\eta(0) = \mathbf{0}$ and $\eta(1) = \mathbf{0}$ so the endpoints remain fixed), the perturbed path is

$$\Gamma_\epsilon(t) \;=\; \Gamma(t) \;+\; \epsilon\,\eta(t).$$

Define the perturbed functional

$$\mathcal{E}(\Gamma_\epsilon) \;=\; \int_0^1 c\big(\Gamma_\epsilon(t)\big)\,\big\|\Gamma_\epsilon'(t)\big\|\,\mathrm{d}t.$$

We say Γ is a minimizer if

$$\frac{\mathrm{d}}{\mathrm{d}\epsilon}\,\mathcal{E}(\Gamma_\epsilon)\bigg|_{\epsilon=0} = 0, \quad \text{for all admissible } \eta(\cdot). \tag{10}$$

Hence, to find the Euler-Lagrange condition, we expand the integrand to first order in ϵ and set that derivative to zero.

3.2.2 Expanding the Perturbed Functional

Let $c_\epsilon(t) = c\big(\Gamma_\epsilon(t)\big)$ and $\mathbf{v}_\epsilon(t) = \Gamma'_\epsilon(t)$. Then the perturbed energy functional is

$$\mathcal{E}(\Gamma_\epsilon) = \int_0^1 c_\epsilon(t)\,\|\mathbf{v}_\epsilon(t)\|\,\mathrm{d}t.$$

We wish to compute the derivative of this integral with respect to ϵ, specifically at $\epsilon = 0$. By applying the standard rule for interchanging differentiation and integration (sometimes called the Leibniz rule), we get:

$$\frac{\mathrm{d}}{\mathrm{d}\epsilon}\Big[\int_0^1 c_\epsilon(t)\,\|\mathbf{v}_\epsilon(t)\|\,\mathrm{d}t\Big] = \int_0^1 \frac{\partial}{\partial\epsilon}\Big(c_\epsilon(t)\,\|\mathbf{v}_\epsilon(t)\|\Big)\,\mathrm{d}t.$$

Next, we use the product rule for differentiation inside the integrand:

$$\frac{\partial}{\partial\epsilon}\Big(c_\epsilon(t)\,\|\mathbf{v}_\epsilon(t)\|\Big) = \frac{\partial c_\epsilon}{\partial\epsilon}(t)\,\|\mathbf{v}_\epsilon(t)\| + c_\epsilon(t)\,\frac{\partial}{\partial\epsilon}\Big[\|\mathbf{v}_\epsilon(t)\|\Big].$$

Hence,

$$\frac{\mathrm{d}}{\mathrm{d}\epsilon}\Big[\int_0^1 c_\epsilon(t)\,\|\mathbf{v}_\epsilon(t)\|\,\mathrm{d}t\Big] = \int_0^1 \Big\{\frac{\partial c_\epsilon}{\partial\epsilon}(t)\,\|\mathbf{v}_\epsilon(t)\| + c_\epsilon(t)\,\frac{\partial}{\partial\epsilon}\Big[\|\mathbf{v}_\epsilon(t)\|\Big]\Big\}\,\mathrm{d}t.$$

Term 1: Variation of $c_\epsilon(t)$. Recall that $c_\epsilon(t) = c\big(\Gamma_\epsilon(t)\big)$, where $\Gamma_\epsilon(t) = \Gamma(t) + \epsilon\,\eta(t)$. Then, by the chain rule,

$$\frac{\partial c_\epsilon}{\partial\epsilon}(t) = \nabla c\big(\Gamma_\epsilon(t)\big)\cdot\frac{\partial\Gamma_\epsilon(t)}{\partial\epsilon} = \nabla c\big(\Gamma_\epsilon(t)\big)\cdot\eta(t).$$

Evaluating at $\epsilon = 0$, $\Gamma_\epsilon(t)$ coincides with $\Gamma(t)$, so we often write

$$\frac{\partial c_\epsilon}{\partial\epsilon}(t)\Big|_{\epsilon=0} = \nabla c\big(\Gamma(t)\big)\cdot\eta(t).$$

Intuitively, this captures how the local cost $c(\mathbf{x})$ changes if we infinitesimally nudge the curve $\Gamma(t)$ in the direction of $\eta(t)$.

Term 2: Variation of $\|\mathbf{v}_\epsilon\| = \|\Gamma'_\epsilon(t)\|$. We set $\mathbf{v}_\epsilon(t) = \Gamma'_\epsilon(t)$. In particular,

$$\mathbf{v}_\epsilon(t) = \frac{\mathrm{d}}{\mathrm{d}t}\Big[\Gamma(t) + \epsilon\,\eta(t)\Big] = \Gamma'(t) + \epsilon\,\eta'(t).$$

Thus, applying the chain rule again to the norm $\|\mathbf{v}_\epsilon\|$,

$$\frac{\partial}{\partial\epsilon}\big[\|\mathbf{v}_\epsilon(t)\|\big] = \frac{\mathbf{v}_\epsilon(t)\cdot\frac{\mathrm{d}}{\mathrm{d}\epsilon}\mathbf{v}_\epsilon(t)}{\|\mathbf{v}_\epsilon(t)\|}.$$

Evaluating at $\epsilon = 0$ yields

$$\frac{\partial}{\partial\epsilon}\big[\|\mathbf{v}_\epsilon(t)\|\big]\Big|_{\epsilon=0} = \frac{\Gamma'(t)\cdot\eta'(t)}{\|\Gamma'(t)\|},$$

since $\mathbf{v}_\epsilon(t)\big|_{\epsilon=0} = \Gamma'(t)$ and $\frac{\mathrm{d}}{\mathrm{d}\epsilon}\mathbf{v}_\epsilon(t)\big|_{\epsilon=0} = \eta'(t)$.

Putting these pieces together. Combining the two variation terms inside the integral, we obtain:

$$\frac{\mathrm{d}}{\mathrm{d}\epsilon}\mathcal{E}(\Gamma_\epsilon)\Big|_{\epsilon=0} = \int_0^1 \Big\{\underbrace{\nabla c\big(\Gamma(t)\big)\cdot\eta(t)\,\|\Gamma'(t)\|}_{\text{variation of } c_\epsilon(t)} + \underbrace{c\big(\Gamma(t)\big)\,\frac{\Gamma'(t)\cdot\eta'(t)}{\|\Gamma'(t)\|}}_{\text{variation of } \|\mathbf{v}_\epsilon\|}\Big\}\,\mathrm{d}t. \tag{11}$$

Thus, the first-order change in $\mathcal{E}(\Gamma_\epsilon)$ with respect to ϵ splits neatly into:

$$\text{(change in cost function } c_\epsilon) + \text{(change in the speed factor } \|\Gamma'(t)\|).$$

This expression forms the core of the Euler-Lagrange derivation. We next set it to zero for all admissible $\eta(t)$ to obtain the stationarity condition governing Γ.

3.2.3 Integrating the Second Part by Parts

Focus on

$$\int_0^1 c\big(\Gamma(t)\big)\,\frac{\Gamma'(t)\cdot\eta'(t)}{\|\Gamma'(t)\|}\,\mathrm{d}t.$$

We can rewrite $\Gamma'(t)\cdot\eta'(t)$ via an integration by parts (and noting $\eta(0)=\eta(1)=0$):

$$\int_0^1 \Gamma'(t)\,\cdot\,\eta'(t)\,\mathrm{d}t = \big[\Gamma(t)\cdot\eta(t)\big]_0^1 \;-\; \int_0^1 \Gamma''(t)\,\cdot\,\eta(t)\,\mathrm{d}t \;=\; -\int_0^1 \Gamma''(t)\,\cdot\,\eta(t)\,\mathrm{d}t.$$

Hence the second term becomes

$$\int_0^1 c\big(\Gamma(t)\big)\,\frac{\Gamma'(t)\cdot\eta'(t)}{\|\Gamma'(t)\|}\,\mathrm{d}t = -\int_0^1 c\big(\Gamma(t)\big)\,\frac{\Gamma''(t)}{\|\Gamma'(t)\|}\,\cdot\,\eta(t)\,\mathrm{d}t.$$

Substitute this back into (11), grouping all dependence on $\eta(t)$, we obtain:

$$\frac{\mathrm{d}}{\mathrm{d}\epsilon}\mathcal{E}(\Gamma_\epsilon)\bigg|_{\epsilon=0} = \int_0^1 \Big[\nabla c\big(\Gamma(t)\big)\,\cdot\,\Gamma'(t)\;-\;c\big(\Gamma(t)\big)\,\frac{\Gamma''(t)}{\|\Gamma'(t)\|}\Big]\frac{\eta(t)}{\|\Gamma'(t)\|}\,\mathrm{d}t.$$

In short, the variation must vanish for all $\eta(t)$, so the bracketed expression must be zero:

$$\nabla c\big(\Gamma(t)\big)\,\cdot\,\Gamma'(t)\;-\;c\big(\Gamma(t)\big)\,\frac{\Gamma''(t)}{\|\Gamma'(t)\|}\;=\;0,\quad\forall t. \tag{12}$$

Geometrically, it indicates that $\Gamma''(t)$ (the local curvature direction) and $\nabla c(\Gamma(t))$ (the local cost gradient) must balance. Rewriting this in a geometrical PDE often yields $\|\nabla T\| = c$.

3.3 Approach II: Value Function and the Eikonal PDE

3.3.1 Definition of $T(\mathbf{x})$

We define a scalar function $T:\Omega\to\mathbb{R}$ by:

$$T(\mathbf{x})\;=\;\min_{\Gamma:\,\Gamma(0)=\mathbf{s},\,\Gamma(1)=\mathbf{x}}\int_0^1 c(\Gamma(t))\,\|\Gamma'(t)\|\,\mathrm{d}t. \tag{13}$$

Hence $T(\mathbf{x})$ is the minimal path cost from $\mathbf{s}$ to $\mathbf{x}$, and in particular $T(\mathbf{s})=0$.

3.3.2 Infinitesimal Consistency of T

If T is truly the minimal cost function, then traveling from $\mathbf{x}$ to its neighbor $\mathbf{x}+\mathrm{d}\mathbf{x}$ must add a small cost $\mathrm{d}T\approx c(\mathbf{x})\,\|\mathrm{d}\mathbf{x}\|$. Meanwhile, from standard differential calculus:

$$\mathrm{d}T = \nabla T(\mathbf{x})\;\cdot\;\mathrm{d}\mathbf{x} = \|\nabla T(\mathbf{x})\|\;\|\mathrm{d}\mathbf{x}\|\;\cos(\theta),$$

where θ is the angle between $\nabla T(\mathbf{x})$ and $\mathrm{d}\mathbf{x}$. For $\mathrm{d}T$ to equal $c(\mathbf{x})\,\|\mathrm{d}\mathbf{x}\|$ in the optimal direction, the angle θ must be zero (or π), thus:

$$\|\nabla T(\mathbf{x})\|\;=\;c(\mathbf{x}).$$

This is precisely the **Eikonal equation**, which we write in final form:

$$\|\nabla T(\mathbf{x})\| = c(\mathbf{x}),\quad T(\mathbf{s})=0. \tag{14}$$

Geometrically, $\|\nabla T(\mathbf{x})\|$ is the rate of change of T in the steepest direction, and $c(\mathbf{x})$ is the local cost "speed." Requiring $\mathrm{d}T = c(\mathbf{x})\,\mathrm{d}s$ gives $\|\nabla T\| = c$.

Key Boundary Condition. Since $T(\mathbf{s})=0$ must hold, the PDE (14) plus that Dirichlet condition at $\mathbf{s}$ uniquely determines T throughout Ω, assuming $c(\mathbf{x})>0$ except in obstacles where $c(\mathbf{x})$ is effectively infinite.

3.4 Connection of Both Approaches

Both the parametric Euler-Lagrange approach (Approach I) and the value-function approach (Approach II) yield the same essential PDE. The first approach more directly shows how curvature, normal direction, and potential gradient (the gradient of c) interplay in the minimal path problem. The second approach makes the PDE (14) more explicit and is typically used for implementation, as the characteristic "wavefront" from $\mathbf{s}$ to the rest of Ω can be tracked by *fast marching* or *fast sweeping* schemes.

3.5 Resulting Gradient Flow and Final PDE Statement

Putting it all together, the PDE we must solve to obtain the minimal-cost value function is:

$$\|\nabla T(\mathbf{x})\| \;=\; c(\mathbf{x}), \quad T(\mathbf{s}) = 0. \tag{15}$$

Any path from $\mathbf{s}$ to $\mathbf{g}$ that *follows the gradient of T in reverse* will realize exactly cost $T(\mathbf{g})$, which is the global minimum. Concretely, the minimal path $\Gamma^*(t)$ is found by solving

$$\frac{\mathrm{d}}{\mathrm{d}t}\Gamma^*(t) \;=\; -\frac{\nabla T(\Gamma^*(t))}{\|\nabla T(\Gamma^*(t))\|}, \quad \Gamma^*(1) = \mathbf{g},$$

i.e. we start at $\mathbf{g}$ and follow $-\nabla T$ until reaching $\mathbf{s}$ (where $T = 0$).

Having derived (15) in detail, the next step is **discretizing and numerically solving** it, which we do (via a fast marching or fast sweeping scheme) in Section 4.

4 Numerical Discretization and Implementation

In the previous section, we derived the Eikonal PDE

$$\|\nabla T(\mathbf{x})\| \;=\; c(\mathbf{x}), \quad T(\mathbf{s}) = 0, \tag{16}$$

as the condition ensuring that $T(\mathbf{x})$ captures the minimal path cost from a start location $\mathbf{s}$ to any point $\mathbf{x}$. We now address *how to discretize and implement* this PDE on a computer. Because (16) is a first-order, non-linear PDE, we must pay particular attention to the directional dependence (so-called *upwind* effects). In this section, we will:

- Show how to build a spatial finite-difference approximation of $\|\nabla T\|$ that correctly accounts for upwinding.
- Justify the need for a single-pass method (Fast Marching) versus naive iterative schemes.
- Derive and explain the local update formula for $T(\mathbf{x})$ that enforces $\|\nabla T\| = c(\mathbf{x})$.
- Discuss why we cannot simply use central differences or straightforward smoothing for a PDE that has strong directional (upwind) character.

4.1 Spatial Discretization: Upwind Approximation of ∇T

4.1.1 Motivation for an Upwind Scheme

Recall that the Eikonal PDE states

$$\|\nabla T(\mathbf{x})\| \;=\; c(\mathbf{x}), \quad T(\mathbf{s}) = 0.$$

In one dimension, ∇T reduces to a derivative $T'(x)$, and the condition $|T'(x)| = c(x)$ must be solved in the direction consistent with increasing x if we know T on the left, or decreasing x if we know T from the right.

For two (or three) dimensions, the gradient ∇T has multi-directional upwind structure. Specifically, $T(\mathbf{x})$ is monotonically built up from smaller T-values near the source $\mathbf{s}$ to larger T-values away from $\mathbf{s}$. Hence, central differencing would incorrectly blend future (unknown) information into the current node, causing convergence or stability issues. Upwind finite differences, in contrast, ensure that $T(\mathbf{x})$ depends only on neighboring nodes whose values are already smaller (and thus physically correct in a wavefront sense).

4.1.2 Godunov or L^∞–type Formulation in 1D

To illustrate the key idea, consider the 1D Eikonal PDE

$$|\,T'(x)\,| \;=\; c(x),$$

where $T'(x)$ is the derivative of T. Locally, we can split $T'(x)$ into

$$T'(x) \;\approx\; \begin{cases} \dfrac{T(x) - T(x-h)}{h}, & \text{if the solution grows from left to right,} \\[2ex] \dfrac{T(x+h) - T(x)}{h}, & \text{if the solution is computed from the right.} \end{cases}$$

In a simpler PDE with $\partial T/\partial t + f(T)\,T'(x) = 0$, we would choose the appropriate upwind difference based on the sign of $f(T)$. Here, in the Eikonal setting, we do not have a flux function $f(\cdot)$ but do know that T grows outward from the source. Thus, the upwind direction is from smaller T toward bigger T. In multiple dimensions, we handle partial derivatives with a max–min or Godunov approach that picks forward or backward difference according to the known or accepted T values.

4.1.3 Godunov's Scheme for 2D

In two dimensions, the PDE $\|\nabla T\| = c$ means

$$\sqrt{\left(\partial T/\partial x\right)^2 + \left(\partial T/\partial y\right)^2} \;=\; c(x,y).$$

A well-known approach (following Tsitsiklis, Osher–Sethian, etc.) uses *upwind differences* for $\partial T/\partial x$ and $\partial T/\partial y$:

$$\begin{aligned} &\max\Big(D_x^- T_{i,j},\, 0\Big)^2 \;+\; \min\Big(D_x^+ T_{i,j},\, 0\Big)^2 \quad \text{(for x-direction)},\\ &\max\Big(D_y^- T_{i,j},\, 0\Big)^2 \;+\; \min\Big(D_y^+ T_{i,j},\, 0\Big)^2 \quad \text{(for y-direction)}, \end{aligned} \tag{17}$$

where $D_x^- T_{i,j} \approx \frac{T_{i,j} - T_{i,j-1}}{\Delta x}$ is the backward difference, $D_x^+ T_{i,j} \approx \frac{T_{i,j+1} - T_{i,j}}{\Delta x}$ is the forward difference, and similarly for $D_y^{\pm}$. Intuitively, this scheme picks the correct directional derivative for each dimension so that information propagates from cells with smaller T-values to cells with larger T-values.

Then, imposing

$$\Big(\text{Godunov upwind approx of } \partial_x T\Big)^2 \;+\; \Big(\text{Godunov upwind approx of } \partial_y T\Big)^2 \;=\; c_{i,j}^2$$

gives a nonlinear update equation for $T_{i,j}$. One typical local solution approach (for 2D Eikonal) is the quadratic formula trick used in the FAST SWEEPING or FAST MARCHING methods:

$$\begin{aligned} &\text{Suppose } T_{i,j-1}, T_{i,j+1}, T_{i-1,j}, T_{i+1,j} \text{ are known.}\\ &\text{Then let } a = \min(T_{i,j-1},\, T_{i,j+1}), \quad b = \min(T_{i-1,j},\, T_{i+1,j}),\\ &\text{and solve } \max\{(T_{i,j} - a),\, 0\}^2 \;+\; \max\{(T_{i,j} - b),\, 0\}^2 \;=\; (c_{i,j}\,\Delta x)^2 \end{aligned}$$

for $T_{i,j}$. Concretely, one obtains a formula like

$$\begin{aligned} T_{i,j} &= \frac{a+b}{2} + \sqrt{\frac{(a+b)^2}{4} - (a^2 + b^2 - c_{i,j}^2\,\Delta x^2)/2}\,,\\ &\text{subject to } T_{i,j} \geq \max\{a,b\} \text{ (otherwise pick } T_{i,j} = \min(a,b) + c_{i,j}\,\Delta x). \end{aligned} \tag{18}$$

A correct derivation ensures that $T_{i,j}$ never depends on unknown future values, only on smaller (already accepted) values.

4.2 Single-Pass Fast Marching Implementation

4.2.1 Why Not Simple Iteration?

One might attempt a time-marching approach for $\partial T/\partial t = c - \|\nabla T\|$, or an iterative approach that relaxes T until $\|\nabla T\| = c$ is reached. However, $\|\nabla T\| = c$ is a *nonlinear, first-order* PDE that can lead to very slow convergence or artifacts if we simply apply naive Gauss–Seidel iteration. The Fast Marching method (Sethian, Tsitsiklis) resolves this by processing grid points in an order consistent with wavefront propagation, guaranteeing each point's T is accepted once without further updates, in a single pass.

4.2.2 Data Structures: Trial and Accepted States

We partition the domain into:

- **Accepted nodes:** where T is final (smallest possible).
- **Trial nodes:** neighbors of accepted nodes, with a temporary T that may still be lowered.
- **Far nodes:** nodes not yet close to the wavefront, $T = \infty$ initially.

We maintain a priority queue (min-heap) of trial nodes keyed by T. Initially, only $\mathbf{s}$ is accepted ($T(\mathbf{s}) = 0$), and all others are far. Then:

1. **Pop** the trial node with the smallest T. Mark it accepted.
2. **Update** its upwind neighbors, applying the local Eikonal solve (18).
3. If a neighbor's new T is smaller than before, push or update that neighbor in the priority queue.
4. Repeat until the goal $\mathbf{g}$ is accepted or the queue is empty.

Since each node's final T is "locked in" the moment it is popped from the queue (it cannot be lowered further), we achieve a single-pass solution that is $\mathcal{O}(N \log N)$ for an N-node grid. The monotonic expansion property ensures correctness.

4.2.3 Why Upwind is Crucial and Why Not Central Differences?

Central differences would mix $T_{i+1,j}$ and $T_{i-1,j}$ symmetrically. But the Eikonal PDE is truly a one-way wave expansion from $\mathbf{s}$. An upwind scheme ensures we only consult T at nodes where the wave has already arrived. Central differencing would incorporate "future" (not yet accepted) values, corrupting the solution and possibly causing cyclical updates.

4.2.4 Justification of Local Eikonal Update

The local update formula (18) can be derived by squaring the PDE $\sqrt{(T_x)^2 + (T_y)^2} = c$, writing $T_{i,j} - a$ as a discrete approximation of $\partial_x T$, etc. Solving the resulting piecewise quadratic in $T_{i,j}$ yields the minimal positive root that is larger than $\max\{a, b\}$. If that root is below $\max\{a, b\}$, we fallback to $\min(a, b) + c\,\Delta x$. This ensures no spurious solutions arise from the branch of the quadratic formula. Hence the upwind criterion is automatically enforced in the local solve.

4.3 CFL Considerations and Time Step Discussion

While Fast Marching does not explicitly evolve in "time," one can interpret an auxiliary equation:

$$\frac{\partial T}{\partial \tau} \;=\; \max\bigl(0,\; c(\mathbf{x}) - \|\nabla T\|\bigr),$$

which drives $T(\mathbf{x})$ until $\|\nabla T\| = c$ is satisfied, ensuring a non-negative update in the direction of correctness. For such iterative PDE schemes, we would pick a $\Delta\tau$ obeying a CFL condition based on the speed $c(\mathbf{x})$ and the mesh spacing Δx. Specifically,

$$c_{\max}\,\Delta\tau \;\le\; \Delta x \quad \text{(in 1D)}, \qquad c_{\max}\,\Delta\tau \;\le\; \frac{\Delta x}{\sqrt{d}} \quad \text{(in } d \text{ dimensions)}.$$

Failure to respect this results in over-stepping the domain of dependence. In practice, many prefer Fast Marching exactly because it bypasses the repeated iteration or delicate CFL constraints, computing T in a single monotonically expanding front.

4.4 Summary of Discretization Choices and Rationale

Upwind Differences. We choose these for ∇T because the PDE is fundamentally monotonic in the sense of wave expansion. Central differences would fail to converge or would converge to an incorrect solution. Upwind ensures the flow of information is consistent with the PDE's causality.

Local Eikonal Update. Using the piecewise quadratic formula for $\max((T_{i,j}-a)^2, 0)+\max((T_{i,j}-b)^2, 0) = (c_{i,j}\Delta x)^2$ yields an explicit expression for $T_{i,j}$. This is essential for the single-pass Fast Marching method to finalize each node's T in strict ascending order.

Fast Marching vs. Iterative. A naive iterative approach to $\|\nabla T\| = c$ may converge very slowly and requires a careful CFL condition on the time step. In contrast, Fast Marching uses a priority queue to accept nodes in ascending T order only once, guaranteeing $\mathcal{O}(N \log N)$ performance and exact upwind consistency.

CFL or Single-pass. If one does attempt a time-dependent PDE approach $\partial T/\partial \tau = c - \|\nabla T\|$, a stable explicit scheme would need $\Delta\tau$ to respect $c_{\max}\Delta\tau \leq \Delta x$. Instead, by using a direct local solve approach (fast marching), we circumvent repeated global sweeps and the time-step selection altogether.

Hence, the discretization pipeline is:

1. Define a grid with spacing Δx (and Δy if rectangular).
2. Initialize $T(\mathbf{x}) = \infty$ except $T(\mathbf{s}) = 0$.
3. Use upwind differences to form the local Eikonal condition $\|\nabla T\| = c$ at each node.
4. Solve for $T_{i,j}$ from neighbors $T_{i\pm1,j}, T_{i,j\pm1}$ with the formula (18).
5. Manage the acceptance of nodes via a priority queue (fast marching).

This ensures a stable, accurate solution that satisfies the PDE (16) in a single pass.

5 Experiments and Results

In this section, we present our experimental setup and the results of applying our discretized PDE-based path planning in three different environments. We also provide both qualitative visualizations and quantitative analyses to examine the impact of key parameters in our model.

5.1 Experimental Setup

We tested our method on a 2D grid in three distinct environments (Env. A, Env. B, and Env. C). Each environment highlights different challenges, such as obstacle shapes or risk distributions. We use a $10\,\mathrm{x}\,10$ domain, discretized into a $300\,\mathrm{x}\,300$ grid. In all experiments, the start point is set to $(0, 0)$ and the goal point to $(10, 10)$. Our obstacle configurations are as follows:

1. **Environment A:** One large rectangular obstacle.
2. **Environment B:** Multiple circular obstacles of various sizes.
3. **Environment C:** A maze-like layout generated via depth-first search (DFS).

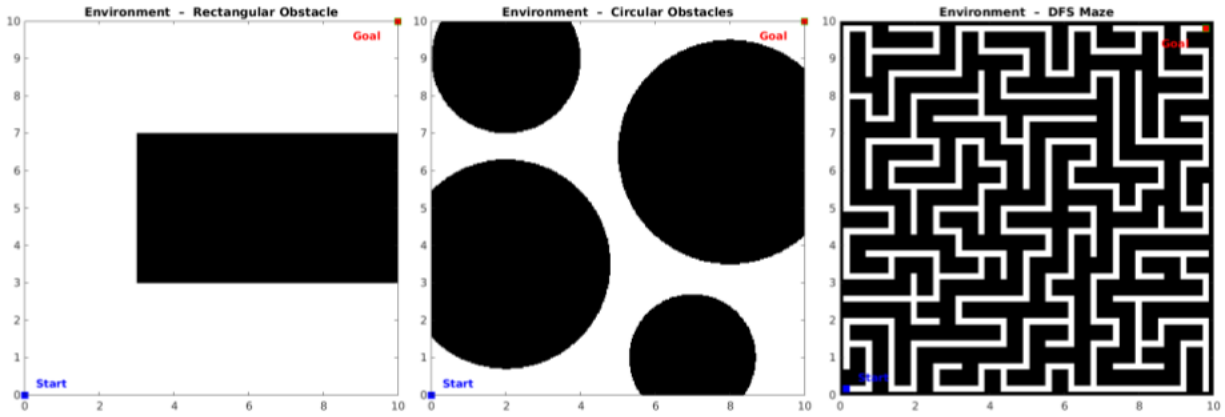


Figure 1: Combined view of the three custom environments (from left to right, Env. A, Env. B, and Env. C). Each environment features different obstacle arrangements and structures to test the robustness of our path planning algorithm.

We used a uniform grid spacing of $\Delta x = 1$ and defined the cost function $c(\mathbf{x})$ as in Section 2, with parameters $\alpha, \beta > 0$. For each environment, we generated the obstacle layout in `MATLAB`, then ran a Fast Marching solver to compute the value function $T(\mathbf{x})$. Finally, we extracted the minimal-cost path by tracing the gradient of T backward from the goal $\mathbf{g}$ to the start $\mathbf{s}$.

5.2 Experimental Results

Table 1: Experimental results for $\alpha = 5.0$ and $\beta = 0.3$ in each environment.

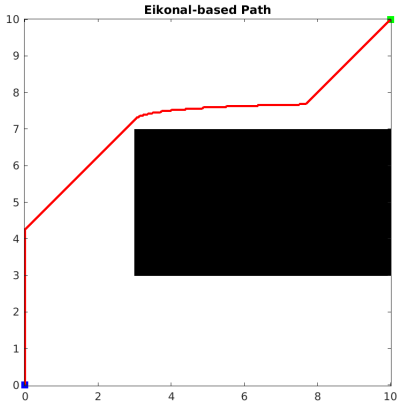
Environment	Final Cost	Path Length	Runtime (s)
A (Rect Obstacle)	504.8196	16.6303	4.5928
B (Circular Obstacles)	574.2850	18.3302	4.2220
C (Maze)	4596.2319	47.6869	3.7594

We tested the PDE-based path planning with $\alpha = 5.0$ and $\beta = 0.3$ on three different environments (A, B, C). Table 1 shows the final cost $T(\mathbf{g})$, the Euclidean path length of the solution, and the overall runtime

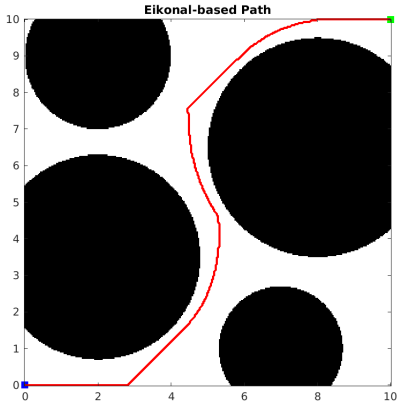
in seconds for each environment. We observe that Environment C (the maze) yields a significantly higher final cost and path length, reflecting its more complex, winding corridors.

From these results, we see that Environment A and B require only moderate cost and length, whereas Environment C's complex maze structure forces a much longer route (nearly three times the path length of A or B). In spite of the increased path complexity, the runtime remains within a few seconds for all environments.

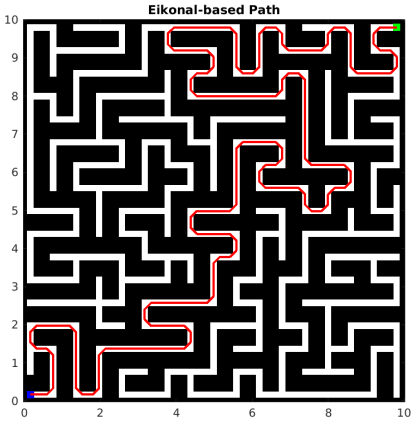
From these numerical results, we can also visualize the actual paths in each environment. Figure 2 displays the minimal-cost solutions superimposed on the obstacle maps:



(a) Environment A



(b) Environment B



(c) Environment C

Figure 2: Minimal-cost paths for $\alpha = 5, \beta = 0.3$ in each environment. The start (blue), goal (green), and resulting path (red) are shown. Environments A and B are displayed side by side (top), while Environment C is shown below (bottom).

5.3 Qualitative Analysis

To visualize how different parameter choices affect the final path, we compare two (α, β) combinations, namely (5, 0.3) and (2, 1.0), on all three environments. Figure 3 shows these paths side by side:

- **Top row:** Rectangular obstacle (Environment A).
- **Middle row:** Multiple circular obstacles (Environment B).

- **Bottom row:** Maze layout (Environment C).

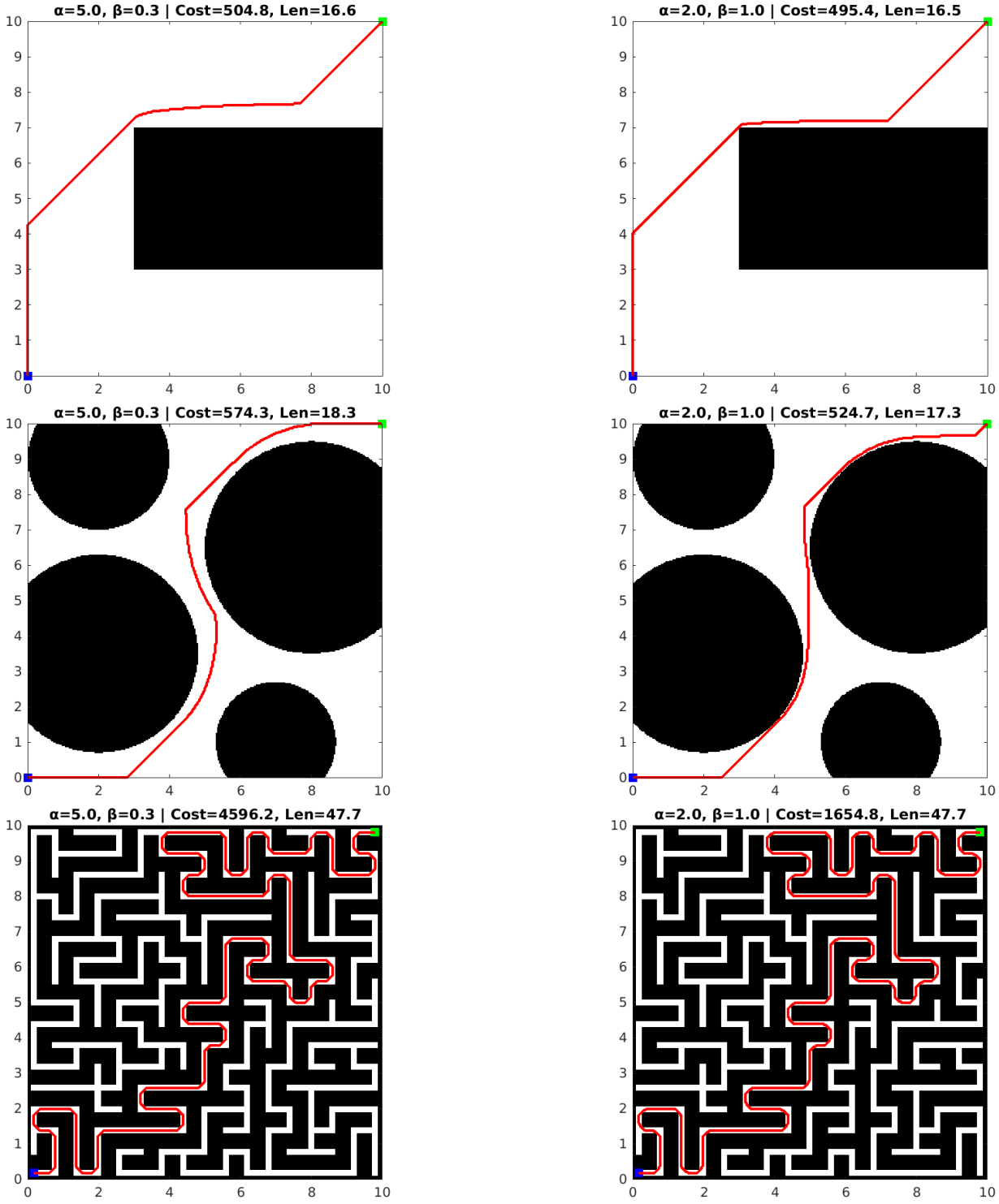


Figure 3: Qualitative paths for two parameter sets (α, β) in each of the three environments. Left: $\alpha = 5, \beta = 0.3$. Right: $\alpha = 2, \beta = 1.0$. The reported cost (Cost=·) and path length (Len=·) are shown in each sub-plot's title.

In the rectangular obstacle case (top row), both parameter sets produce nearly direct routes along the upper boundary, but the cost is slightly higher when $\alpha = 5$ and $\beta = 0.3$ because the risk penalty accumulates more strongly near the obstacle. In contrast, the $(\alpha = 2, \beta = 1.0)$ route has a modestly lower cost and a comparable path length.

For the circular obstacles (middle row), $\alpha = 5, \beta = 0.3$ again yields a path that keeps a bit more distance from the outer obstacles, whereas (2, 1.0) allows the path to "hug" the circular boundary more tightly. This manifests in a slightly shorter total path length but also a moderately lower total cost.

Finally, in the maze environment (bottom row), both parameter sets result in similarly winding routes through the narrow corridors, and the path lengths end up quite close (both near 47–48). However, the cost for $\alpha = 5, \beta = 0.3$ is significantly larger, reflecting how the cost function penalizes extended travel near walls.

Overall, these qualitative comparisons illustrate how increasing α (or decreasing β) tends to inflate the cost near obstacles, thus either lengthening the route or accumulating more penalty. Conversely, lower α or higher β reduces the local penalty more quickly, often leading to a route that hugs the obstacle boundary more closely or yields a lower total cost.

5.4 Quantitative Analysis

We tested four values of $\alpha \in \{2, 5, 7, 9\}$ and four values of $\beta \in \{0.3, 0.5, 0.7, 0.9\}$ in each of the three environments, yielding 16 parameter combinations per environment. Tables 2-4 show the numerical results for final cost ($T(\mathbf{g})$), path length, and runtime (in seconds).

From these tables, we see that α and β have varying degrees of influence on the final path. In particular:

- **Environment A (Rectangular Obstacle):** Cost and length remain in the 495–507 and 16.4–16.7 range, showing that this obstacle layout does not force dramatic changes. Higher α slightly increases the cost but does not significantly affect path length.
- **Environment B (Multiple Circular Obstacles):** The cost spans about 526–587, reflecting how greater α generally raises the penalty near obstacles. Meanwhile, path length hovers around 17–18.5.
- **Environment C (Maze):** Costs vary dramatically, from about 1709 up to 7082, illustrating how sensitive the maze environment is to penalty definitions. Larger α and smaller β heavily penalize hugging the walls, leading to extremely high total cost (even though the path length does not vary as much).

Table 2: Quantitative results for **Environment A**.

α	β	**Cost**	**Length**	**Runtime**
2.0	0.3	501.92	16.53	4.8229
2.0	0.5	498.17	16.51	4.6509
2.0	0.7	496.59	16.51	4.5540
2.0	0.9	495.71	16.49	4.5291
5.0	0.3	504.82	16.63	4.7848
5.0	0.5	499.92	16.57	4.7236
5.0	0.7	497.82	16.55	4.6015
5.0	0.9	496.65	16.53	4.6164
7.0	0.3	505.86	16.67	4.9242
7.0	0.5	500.56	16.61	4.7563
7.0	0.7	498.29	16.57	4.5780
7.0	0.9	497.00	16.55	4.5076
9.0	0.3	506.63	16.71	4.8766
9.0	0.5	501.01	16.61	4.6945
9.0	0.7	498.59	16.57	4.5786
9.0	0.9	497.26	16.55	4.6424

Table 3: Quantitative results for **Environment B**.

α	β	**Cost**	**Length**	**Runtime**
2.0	0.3	559.74	18.02	4.2778
2.0	0.5	539.52	17.64	4.3226
2.0	0.7	531.20	17.49	4.4322
2.0	0.9	526.49	17.36	4.3819
5.0	0.3	574.29	18.33	4.6233
5.0	0.5	545.88	17.87	4.3918
5.0	0.7	535.56	17.62	4.4770
5.0	0.9	529.90	17.47	4.4377
7.0	0.3	581.10	18.44	4.7233
7.0	0.5	548.17	17.96	4.5443
7.0	0.7	537.28	17.72	4.5606
7.0	0.9	531.07	17.57	4.4891
9.0	0.3	587.07	18.51	4.6280
9.0	0.5	550.01	17.98	4.5042
9.0	0.7	538.39	17.72	4.6427
9.0	0.9	532.05	17.57	4.6561

Table 4: Quantitative results for **Environment C**.

α	β	**Cost**	**Length**	**Runtime**
2.0	0.3	2723.89	47.69	4.6011
2.0	0.5	2185.36	47.69	4.4938
2.0	0.7	1880.24	47.69	4.2073
2.0	0.9	1709.54	47.69	4.1757
5.0	0.3	4596.23	47.69	4.2783
5.0	0.5	3224.64	48.82	4.3007
5.0	0.7	2457.58	49.80	4.2117
5.0	0.9	2035.86	49.80	4.1623
7.0	0.3	5839.08	48.74	4.4003
7.0	0.5	3906.24	49.80	4.4037
7.0	0.7	2832.27	49.80	4.2661
7.0	0.9	2241.91	49.80	5.0139
9.0	0.3	7081.92	48.74	4.2249
9.0	0.5	4587.69	49.80	4.3540
9.0	0.7	3206.96	49.80	5.1961
9.0	0.9	2447.92	49.80	4.9983

In summary, increasing α typically raises the cost of traveling near obstacles, while decreasing β makes that cost persist over longer distances (leading to inflated final cost). The maze environment Env C shows the largest contrast, since it requires extended traversal near walls, making the total penalty very sensitive to α, β.

6 Conclusion

In this project, we applied a PDE-based approach. Specifically, the Eikonal equation with a soft cost function to the safe path planning problem. Although lectures had covered the Eikonal formulation in a theoretical sense, our hands-on experiments revealed additional insights into both its strengths and limitations:

- **Strengths:**
  - The method guarantees a global optimum under the given cost function, avoiding local minima in complex obstacle fields.
  - By adjusting (α, β) in the cost definition, we can easily tune how strongly the path avoids obstacle boundaries, striking a chosen balance between path length and safety.
  - Fast Marching provides an efficient single-pass implementation, yielding stable results in near $\mathcal{O}(N \log N)$ time on a grid of N nodes.
- **Weaknesses:**
  - Our cost model assumes a static environment, so moving obstacles or time-dependent risks are not addressed. Extending the PDE to dynamic or high-dimensional scenarios could become more complex, potentially losing the single-pass advantage.
  - The exponential cost term $1 + \alpha e^{-\beta \,\mathrm{dist}}$ was effective for basic obstacle avoidance, but we saw in the "maze" environment that large α values can inflate the total cost dramatically. This suggests a need for a more refined cost mechanism or additional constraints for corridors.
  - The approach depends heavily on accurate distance-to-obstacle computation (via $\mathrm{dist}(\mathbf{x}, \mathcal{O})$). If the underlying grid resolution is too coarse, we may misjudge the penalty for narrow passages or very close obstacles.

Future Directions. If more time were available, we would consider:

- Exploring anisotropic or direction-dependent costs, allowing, for example, different penalties in forward vs. lateral directions.
- Incorporating dynamic obstacles, requiring a time-dependent PDE framework or repeated updates of $T(\mathbf{x}, t)$.
- Investigating an adaptive or multi-scale grid that refines around obstacles, thus capturing narrow corridors more accurately without uniformly increasing the entire domain resolution.

Overall, the Eikonal PDE approach proved robust for static, continuous-field path planning, offering clear parameter controls (α, β) and a global solution via Fast Marching. Nonetheless, addressing dynamic scenarios and tuning the cost model for highly complex environments remain avenues for further research and refinement.