Log-Space Harmonic Function Path Planning

Kyle Hollins Wray, Dirk Ruiken, Roderic A. Grupen, and Shlomo Zilberstein

Abstract— **We propose a log-space solution for robotic path planning with harmonic functions that solves the long-standing numerical precision problem. We prove that this algorithm: (1) performs the correct computations in log-space, (2) returns the true equivalent path using the log-space mapping, and (3) has a strong error bound given its convergence criterion. We evaluate the algorithm on 7 problem domains. A Graphics Processing Unit (GPU) implementation is also shown to greatly improve performance. We also provide an open source library entitled *epic* with extensive ROS support and demonstrate this method on a real humanoid robot: the uBot-6. Experiments demonstrate that the log-space solution rapidly produces smooth obstacle-avoiding trajectories, and supports planning in exponentially larger real-world robotic applications.**

I. Introduction

Path planning is a critical component of most robotic systems but is quite challenging to perform in real-time robotic scenarios [1]. Applications include autonomous cars [2], smart wheelchairs [3], and unmanned aerial vehicles [4]. A number of techniques have emerged, each with their own benefits and drawbacks. *Rapidly-exploring Random Trees (RRT)* [5] randomly construct trees following any general state transition, including any non-holonomic constraints. *Probabilistic Roadmaps (PRMs)* [6] begin by randomly selecting points to create a collision-free graph. Then, it enters a query phase that connects the start and goal locations. Eventually it finds the fastest route within the final graph. Connolly and Grupen [7] proposed a prominent *potential field method* that employs solutions to *Laplace's equation* called *harmonic functions*. Specifically, these are solutions over a grid of a bounded region. The method produces an optimal smooth flow, avoiding obstacles as the robot moves to goal states (i.e., minimizes "hitting probability"). Importantly, it does not suffer from local minima, unlike other potential field methods [7]. There are two main issues with harmonic functions, both regarding scalability: (1) numerical precision issues, and (2) computational complexity.

Rosell and Iniguez [9] scale harmonic functions using probabilistic cell decomposition, so that only a subset of the cells are sampled and therefore less computation occurs. They reduced the effective number of cells by up to 80% within 2-dimensional $\mathcal{C}$-space. Aarno *et al.* [10] merge PRMs with harmonic functions to alleviate the issue with PRMs (shared by RRT too) failing to find paths through the more narrow spaces between obstacles. Garrido *et al.* [11] use a finite element method, instead of the standard finite difference method, to better handle complex obstacles. Torres-Huitzil *et al.* [12] attempted to address precision problem by incrementally using higher precisions (e.g., converge in single-precision, then use double, etc.). The experiments we present demonstrate simply using higher precision does not solve the problem. Scherer *et al.* [13] successfully implemented harmonic path planning for obstacle avoidance on an autonomous helicopter; however, due to speed and numerical precision issues, they used a very small discretization (i.e., $\langle 128, 128, 64 \rangle$ in size). All of these approaches, however, will still suffer from numerical precision issues (described below) using harmonic functions, even for modestly-sized spaces.

This work was supported in part by NASA (grant NASA-GCT-NNX12AR16A) and the NSF (grant IIS-1405550). Any opinions, findings, and conclusions expressed in this material are those of the author(s) and do not necessarily reflect the views of the NASA or the NSF.

College of Information and Computer Sciences, University of Massachusetts, Amherst, MA 01002, USA. Emails: {wray, ruiken, grupen, shlomo}@cs.umass.edu

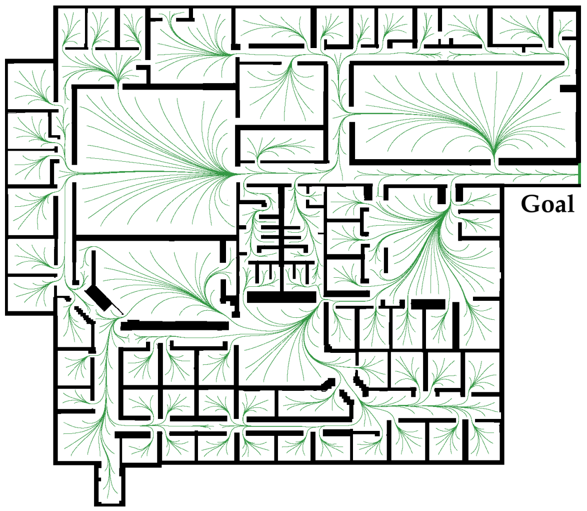

Fig. 1. Harmonic function in Willow Garage's office [8] (1397-by-1213) with obstacles (black) and free space (white). Example streamlines (green) show the log-space Gauss-Seidel GPU implementation's solution. The algorithm is complete, and returns a valid path from all locations.

The key numerical precision issue stems from the common prevalence of obstacles, whose fixed value is 1, and sparsity of goals, whose fixed value is 0. The free spaces average neighbors until convergence; therefore, most of their values are very close to 1, resulting in problems with floating-point addition [14]. The proposed log-space algorithm solves this issue, allowing for *exponentially* larger problem sizes to be solved (Figure 1). Specifically, it exploits the *log-sum-exp* algorithm to perform this near 1 neighbor averaging in a special log-space without precision issues. To the best of our knowledge, no one has proposed this before. Though *log-sum-exp* has been used to solve different issues found in ad hoc network routing with Laplace's equation [15], this is a specific issue and solution for robot path planning.

The primary contributions of this paper are: (1) the log-space algorithm, (2) three propositions regarding its correctness and convergence, (3) a full suite of experiments comparing the algorithm with the original approach, and (4) fully developed Robot Operating System (ROS) support as both a standard *ros_nav* plugin and a custom anytime path planner ROS node, demonstrated on a robot: the uBot-6 [16].

Section II introduces the formal notions of potential fields and harmonic functions. Section III presents the algorithm that maps the problem to log-space. This includes three formal proofs regarding correctness and convergence. Section IV describes a GPU implementation of the log-space algorithm and demonstrates that it allows us to solve exponentially larger problems. Section V concludes with final thoughts.

II. Potential Field Path Planning

Potential field methods describe the path planning problem borrowing notions from physics (e.g., fluid dynamics). Specifically, they assume that all obstacles have a high potential, and the goal state has a low (or simply zero) potential. Solving the corresponding partial differential equation (PDE) will yield a flow which maximizes obstacle avoidance while traveling towards a goal. The trade-off between these two competing objectives produces solutions which instead minimize the probability of hitting an obstacle (i.e., "hitting probability") [17]. The resulting path is smooth and natural.

A. Solutions to Laplace's Equation

We focus on the solutions to Laplace's equation, which are *harmonic functions*. Laplace's equation is a particular form of Poisson's equation with an equality of zero. Formally, we have twice continuously differentiable function $\phi : \Omega \rightarrow \mathbb{R}$ defined on n-dimensional region $\Omega \subset \mathbb{R}^n$ with boundary $\partial\Omega$. Laplace's equation is the partial differential equation (PDE):

$$\bigtriangledown^2 \phi = \sum_{i=1}^{n} \frac{\partial^2 \phi}{\partial \omega_i^2} = 0.$$

The solutions to Laplace's equation are called *harmonic functions*.

Let $\mathbf{m} \in \mathbb{N}^n$ be the dimension sizes for a n-dimensional grid defined as $X = \{\mathbf{x} \in \mathbb{N}^n | \forall i \in \{1, \ldots, n\}, x_i \leqslant m_i\}$. Let grid X correspond to the bounded discrete regular samplings on Ω. Let function $u : X \rightarrow \mathbb{R}$ be defined such that $u(\mathbf{x}) = \phi(\omega)$, $\forall i \in \{1, \ldots, n\}$, such that $\mathbf{x} \in X$ is the corresponding regular sample ("indexes") for $\omega \in \Omega$. We apply Taylor series approximation, use Dirichlet boundary conditions, $\phi(\omega) = 1$ for obstacles, $\phi(\omega) = 0$ for goals, and apply finite (central) difference to solve for each point on Ω sampled following grid X. For brevity, let $O \subseteq X$ be the set of obstacles, and $G \subseteq X$ be the set of goals. This yields:

$$0 = \bigtriangledown^2 \phi(\omega) = \Big(\sum_{i=1}^{n} u(x_i \pm 1, \mathbf{x}_{-i}) \Big) - 2n \cdot u(\mathbf{x})$$

with $\mathbf{x}_{-i} = [x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n]^T$, such that $u(x, \mathbf{x}_{-i})$ denotes $u([x_1, \ldots, x_{i-1}, x, x_{i+1}, \ldots, x_n]^T)$. This produces a system of linear equations with $O(m_1 \cdots m_n)$ unknowns. This is sparse so we use iterative methods such as *Gauss-Seidel* or *Successive Over Relaxation (SOR)*.

SOR operates under a particular ordering of the equations in the system of linear equations (for the finite difference method). It uses values that are computed for the current iteration, providing overlap that greatly reduces the number of iterations, as compared to using simpler orderings. We assume a *red-black ordering*, creating an n-dimensional "checkerboard" pattern for red-labeled cells $R \subset X$ and black-labeled cells $B \subset X$. Each iteration, we first update red cells then black cells, alternating between the two. The benefit to this SOR ordering is that we can perfectly parallelize each update. The update equation at iteration t, for each free space cell $\mathbf{x} \in R$ (t even) or $\mathbf{x} \in B$ (t odd) is:

$$u^{t+1}(\mathbf{x}) = (1 - \lambda) u^t(\mathbf{x}) + \frac{\lambda}{2n} \sum_{i=1}^{n} u^t(x_i \pm 1, \mathbf{x}_{-i}) \qquad (1)$$

with relaxation parameter λ that enables faster convergence with $\lambda > 1$. Boundary cells O and G have fixed values of 1 and 0, respectively. If $\lambda > 1$, it cannot be used as an anytime algorithm because it will produce poor initial solutions due to the relaxation parameter causing oscillations to overshoot the true value. The slightly more "well-behaved" algorithm is known as Gauss-Seidel with $\lambda = 1$.

B. Streamlines

Solving the system of linear equations produces a solution denoted u^*. This allows us to compute *streamlines* for any initial value $\mathbf{s}^0 \in \Omega$ and step size $h \in \mathbb{R}^+$ (Figure 1). At each time step t, we interpolate the values around current point $\mathbf{s}^t$ with the neighboring samples, approximate the derivatives using central difference, and take a small step of size h until a goal (or obstacle) is reached. This is essentially *gradient descent*. Formally, we define a streamline as an ordered list $\mathbf{s} = \langle \mathbf{s}^0, \ldots, \mathbf{s}^\tau \rangle$, assuming it takes τ steps to reach a terminating point. For each $t \in \{0, \ldots, \tau - 1\}$, the location of $\mathbf{s}^{t+1} \in \Omega$ is computed following:

$$\mathbf{s}^{t+1} = \mathbf{s}^t - h \bigtriangledown \hat{\phi}(\mathbf{s}^t) \qquad (2)$$

with $\hat{\phi} : \Omega \rightarrow \mathbb{R}$ as the interpolated approximation of ϕ using u^*, assuming it is normalized to a unit vector.

C. Benefits of Harmonic Function Planners

Harmonic functions are highly useful for path and motion planning in robotics for a number of reasons. First, they are complete (up to the grid resolution) and always find a path from *any location* (not just a single initial location) to the goal(s); there are no local minima [7]. This is important since actual execution via a *path follower* does not perfectly follow the path. Deviations can easily be corrected because there is a valid gradient everywhere, as opposed to other path planners. Second, they obviously handle multiple goals, and obstacles, both in any possible shape as well. Third, their resulting streamlines must be smooth curves, which tend to be more easily followable by a path follower [13]. Fourth, Gauss-Seidel and SOR, as described above, are anytime algorithms that can be used at (almost) anytime to find a path; additional iterations simply improve the streamline. Fifth, it optimally minimizes the probability the robot will hit an obstacle before reaching the goal.

There are, however, some drawbacks to harmonic functions. First, the solution is only as good as the discretization or resolution used to define X in proportion to the size of Ω. This is an issue for most methods which use occupancy grids. These grid precision issues can be addressed by increasing the resolution. Second, it is computationally more expensive when compared to sampling methods such as RRT or PRMs. The first and second issues can be addressed by algorithmic techniques (e.g., *multi-grid* methods), more powerful hardware, or the parallelization techniques we leverage here. Third, perhaps the most crippling drawback is that implementations of Jacobi iteration, Gauss-Seidel, or SOR fail to find valid gradients for all but the smallest grid sizes (e.g., 256-by-256) due to *numerical precision*. In the next section, we solve this well-known issue using a log-space mapping which perfectly preserves all of the harmonic function properties.

III. Log-Space Harmonic Functions

All of the research on potential fields with harmonic functions have limitations due to numerical precision. The solutions returned by any of these iterative methods actually yield incorrect results with even a modest number of cells (e.g., larger than a 256-by-256 grid X with complex boundaries). IEEE floating-point standards [14] for single precision (floats; 4 bytes) have numerical issues in addition/multiplication around $1e^{-7}$. Double precision (doubles; 8 bytes) have numerical these numerical issues around $1e^{-16}$, and even higher precisions (e.g., long doubles; 16 bytes) fail around $1e^{-19}$. Since almost all the neighbor averaging that takes place are near the obstacle value of 1, we quickly run into this issue. The resultant streamlines fail to find the goal from "distant" starting points. Simply multiplying the obstacle potential by a large value (e.g., $1e^{10}$) or changing the precision (e.g., long double) might help provide slightly more "numerical room" for the algorithm; however, this does not address the fundamental scalability issue. In practice, it only helps a very small amount (e.g., Figure 2). To solve this issue, we perform operations in a special log-space.

A. Log-Space Gauss-Seidel Mapping

The log-space mapping works by exploiting the properties of logarithms and exponentiation, namely using the *log-sum-exp* algorithm, denoted ℓse below. This algorithm is commonly used in machine learning to multiply thousands of near-zero probabilities together. It is formally defined for any values $\mathbf{y} = [y_1, \ldots, y_k]^T$ as:

$$\ell se(\mathbf{y}) = y^* + \log \sum_{i=1}^{k} e^{y_i - y^*} \tag{3}$$

with $y^* = \max_i y_i$ in most cases. In order to use this, however, we need to manipulate the numerical relaxation problem into this form.

First, in the log-space Gauss-Seidel algorithm, an auxiliary variable $v : \mathbb{R}^n \to \mathbb{R}$ is introduced for u instead:

$$v(\mathbf{x}) = \log \Big((1 - u(\mathbf{x}))(1 - \delta) + \delta \Big) \tag{4}$$

with a small $\delta > 0$ that determines how closely we can approximate $\log 0$. This maps u to be within the range of $[\delta, 1]$ and then applies a logarithm. Crucially, u values are flipped so that obstacles are 0 and goals are 1. This maximizes the efficiency of *log-sum-exp* because, in practice, almost all values are nearly 1, causing the numerical precision problems. This is due to the commonly disproportionate ratio of obstacles to goals. Thus, δ is really what pushes this model to have exponentially more "numerical room" in which to operate.

For each free space cell $\mathbf{x} \in R$ (t even) or each $\mathbf{x} \in B$ (t odd), the log-space Gauss-Seidel update for iteration t is defined as:

$$v^{t+1}(\mathbf{x}) = \begin{cases} 0, & \text{if } \mathbf{x} \in G \\ \log \delta, & \text{if } \mathbf{x} \in O \\ \ell se(\mathbf{v}^t) - \log 2n, & \text{otherwise} \end{cases} \tag{5}$$

with $\mathbf{v}^t = \langle v^t(x_1 \pm 1, \mathbf{x}_{-1}), \ldots, v^t(x_n \pm 1, \mathbf{x}_{-n}) \rangle$ and obstacle penalty $\log \delta \ll 0$ (e.g., $\log \delta = -1e^{15}$, which implies $\delta = e^{-1e^{15}}$). This is iterated until at some time t:

$$\|v^t - v^{t-1}\|_\infty = \max_{\mathbf{x}} |v^t(\mathbf{x}) - v^{t-1}(\mathbf{x})| < \epsilon \tag{6}$$

with convergence criterion $\epsilon > 0$, but *with respect to* v. Importantly, we *never* convert v back to u; doing so would undo all this effort. Finally, the astute reader might wonder if we can use SOR. This will be explained in the next section.

B. Theoretical Analysis

We prove three important properties, two covering correctness of the approach, and the third addressing the error upon convergence. To begin, Proposition 1 states that the algorithm is correct, namely that it is a valid log-space mapping of the original Gauss-Seidel algorithm.

Proposition 1 (Correctness): Gauss-Seidel (Equation 1 with $\lambda = 1$) mapped to the log-space (Equation 4) is equivalent to the log-space Gauss-Seidel (Equation 5).

Proof: First, we check the two boundary values. If $\mathbf{x} \in G$, then by Equation 4:

$$v^{t+1}(\mathbf{x}) = \log \Big((1 - 0)(1 - \delta) + \delta \Big) = \log 1 = 0$$

Next, if $\mathbf{x} \in O$, then by Equation 4:

$$v^{t+1}(\mathbf{x}) = \log \Big((1 - 1)(1 - \delta) + \delta \Big) = \log \delta$$

With both boundaries correct, we examine the update. By Equation 4, and then Equation 1 with $\lambda = 1$, we have:

$$v^{t+1}(\mathbf{x}) = \log \Big((1 - \frac{1}{2n} \sum_{i=1}^{n} u^t(x_i \pm 1, \mathbf{x}_{-i}))(1 - \delta) + \delta \Big)$$

Let $1 = \sum_{i=1}^{n} (\frac{1}{2n} + \frac{1}{2n})$, and similarly $\delta = \sum_{i=1}^{n} (\frac{\delta}{2n} + \frac{\delta}{2n})$, then distribute $(1 - \delta)$ to obtain:

$$v^{t+1}(\mathbf{x}) = \log \frac{1}{2n} \sum_{i=1}^{n} ((1 - u^t(x_i \pm 1, \mathbf{x}_{-i}))(1 - \delta) + \delta)$$

By the properties $\log a/b = \log a - \log b$ and $z = e^{\log z}$:

$$\begin{aligned} v^{t+1}(\mathbf{x}) &= \log \sum_{i=1}^{n} e^{v^t(x_i \pm 1, \mathbf{x}_{-i})} - \log 2n \\ &= \ell se(\mathbf{v}^t) - \log 2n \end{aligned}$$

Thus, in all three cases, we are able to take the log-space value from Equation 4, apply the original Gauss-Seidel in Equation 1 (with $\lambda = 1$), and create the log-space Gauss-Seidel in Equation 5. ■

Now that we know this algorithm is correct, we need to compute the policy via streamlines. The issue is that we store the solution in log-space; converting it back and computing streamlines runs into the exact same numerical precision problems again. So, we must guarantee that the gradients used in log-space are equal to those used in normal space. This is proven in Proposition 2.

Proposition 2 (Correctness of Log-Space Streamlines): Let $\mathbf{u}$ and $\mathbf{v}$ be the converged results from Equations 1 (with $\lambda = 1$) and 5, respectively. Let $\mathbf{s}_u$ and $\mathbf{s}_v$ be their resulting streamlines, $\mathbf{s}_u$ following gradient *descent* and $\mathbf{s}_v$ following gradient *ascent*. For any initial $\omega \in \Omega$, $\mathbf{s_u} = \mathbf{s_v}$.

Proof: By Equation 2, for any $t \in \{0, \ldots, \tau - 1\}$:

$$\mathbf{s}^{t+1} = \mathbf{s}^t \pm h \bigtriangledown \hat{\phi}(\mathbf{s}^t)$$

It is sufficient to show that $\bigtriangledown \hat{\phi}_u = - \bigtriangledown \hat{\phi}_v$, i.e., that the gradients (unit vectors) are opposites. Equivalently, we show that their dot product is -1.

$$\bigtriangledown \hat{\phi}_u \cdot \bigtriangledown \hat{\phi}_v = \sum_{i=1}^{n} \frac{\frac{\partial u(\mathbf{x})}{\partial x_i}}{\| \bigtriangledown \hat{\phi}_u \|_2} \frac{\frac{\partial v(\mathbf{x})}{\partial x_i}}{\| \bigtriangledown \hat{\phi}_v \|_2}$$

First, we take $\frac{\partial v(\mathbf{x})}{\partial x_i}$ for any i:

$$\begin{aligned} \frac{\partial v(\mathbf{x})}{\partial x_i} &= \frac{\partial}{\partial x_i} \log \Big((1 - u(\mathbf{x}))(1 - \delta) + \delta \Big) \\ &= \frac{1 - \delta}{(1 - u(\mathbf{x}))(1 - \delta) + \delta} \Big(- \frac{\partial u(\mathbf{x})}{\partial x_i} \Big) \end{aligned}$$

Then, we simplify the elements of $\bigtriangledown \hat{\phi}_v$, such that for any i:

$$\begin{aligned} \frac{\frac{\partial v(\mathbf{x})}{\partial x_i}}{\| \bigtriangledown \hat{\phi}_v \|_2} &= \frac{\Big(\frac{1-\delta}{(1-u(\mathbf{x}))(1-\delta)+\delta} \Big) \Big(- \frac{\partial u(\mathbf{x})}{\partial x_i} \Big)}{\sqrt{\sum_{k=1}^{n} \Big(\frac{1-\delta}{(1-u(\mathbf{x}))(1-\delta)+\delta} \Big)^2 \Big(- \frac{\partial u(\mathbf{x})}{\partial x_k} \Big)^2}} \\ &= \frac{- \frac{\partial u(\mathbf{x})}{\partial x_i}}{\sqrt{\sum_{k=1}^{n} \Big(\frac{\partial u(\mathbf{x})}{\partial x_k} \Big)^2}} = - \frac{\frac{\partial u(\mathbf{x})}{\partial x_i}}{\| \bigtriangledown \hat{\phi}_u \|_2} \end{aligned}$$

This manipulation is only possible because $\delta \in [0, 1]$ and $u(\mathbf{x}) \in [0, 1]$. Finally, we apply this to the original equation:

$$\bigtriangledown \hat{\phi}_u \cdot \bigtriangledown \hat{\phi}_v = - \sum_{i=1}^{n} \frac{\frac{\partial u(\mathbf{x})}{\partial x_i}}{\| \bigtriangledown \hat{\phi}_u \|_2} \frac{\frac{\partial u(\mathbf{x})}{\partial x_i}}{\| \bigtriangledown \hat{\phi}_u \|_2} = -(\bigtriangledown \hat{\phi}_u)^2 = -1$$

Thus, this algorithm returns a gradient equal to the negated original gradient. Therefore, the algorithm's streamlines are identical to the original streamlines. ■

Now that we know both the algorithm and the resultant policy's streamlines are correct, we examine the error of the system with respect to the true normal values u after convergence of v. First, we prove a brief Lemma 1, then provide the complete error bound in Proposition 3.

Lemma 1: If $v(\mathbf{x}) > v'(\mathbf{x})$, then $u(\mathbf{x}) < u'(\mathbf{x})$.

Proof: We begin with $v(\mathbf{x}) > v'(\mathbf{x})$, implying that $\log(1 - u(\mathbf{x}))(1 - \delta) + \delta > \log(1 - u'(\mathbf{x}))(1 - \delta) + \delta$. Thus, $1 - u(\mathbf{x}) > 1 - u'(\mathbf{x})$, and so $u(\mathbf{x}) < u'(\mathbf{x})$. ■

Proposition 3 (Error Bound): Let t be the number of iterations until convergence to within ϵ (Equation 6). Let u^t and u^{t-1} be the *non-log-space* converged values and previous time step, respectively. The error from optimal is:

$$\| u^t - u^{t-1} \|_\infty < \frac{e^\epsilon - 1}{e^\epsilon (1 - \delta)} \tag{7}$$

Proof: Take any $\mathbf{x}$. Without loss of generality, let $v^t(\mathbf{x}) - v^{t-1}(\mathbf{x}) > 0$. By Equation 6:

$$\begin{aligned} \epsilon &> v^t(\mathbf{x}) - v^{t-1}(\mathbf{x}) \\ &= \log((1 - u^t(\mathbf{x}))(1 - \delta) + \delta) \\ &\quad - \log((1 - u^{t-1}(\mathbf{x}))(1 - \delta) + \delta) \\ e^\epsilon &> \frac{(1 - u^t(\mathbf{x}))(1 - \delta) + \delta}{(1 - u^{t-1}(\mathbf{x}))(1 - \delta) + \delta} \\ (1 - u^{t-1}(\mathbf{x}))e^\epsilon &> 1 - u^t(\mathbf{x}) + \frac{\delta}{1 - \delta}(1 - e^\epsilon) \\ u^t - u^{t-1}(\mathbf{x})e^\epsilon &> \frac{1 - e^\epsilon}{1 - \delta} \end{aligned}$$

We multiply u^t by $e^\epsilon > 1$ (because $\epsilon > 0$), multiply both sides by -1, and then apply Lemma 1:

$$0 < u^{t-1}(\mathbf{x}) - u^t(\mathbf{x}) < \frac{e^\epsilon - 1}{e^\epsilon (1 - \delta)}$$

Thus, this true for all $\mathbf{x}$, and so we are done. ■

Interestingly, this error bound approximates the error ϵ if we were to simply use the original Gauss-Seidel; however, ϵ grows much faster than this bound from Equation 7. For example, a modest $\delta = e^{-10}$ and $\epsilon = 0.1$ yields an improved error bound of $0.09517 < \epsilon = 0.1$. They do, however, converge to the same performance as $\epsilon \to 0$.

The reason why we do not use SOR can be found in Proposition 1's proof, provided λ is not omitted. This results in a $\log(1 - \lambda)$ term, which is undefined for any $\lambda \geqslant 1$. Intuitively, log-space SOR would cause values to oscillate into zero or negative numbers, during which one cannot take a logarithm. Thankfully, with $\lambda = 1$, this term does not exist.

C. Parallelization

Both Gauss-Seidel and SOR *can* be parallelized, but are not *perfectly parallelizable* like Jacobi iteration. As described in Section II-A, we must order the updates over the n-dimensional grid in a special way; we use a simple red-black ordering. The first half of the updates use neighbors that are not being updated, with the second half of the updates using these newly updated values. The result are two steps, each having completely disconnected cells.

Algorithm 1 GPU Log-Space Gauss-Seidel Update (2-D)

```
Require: $b_{id}$, $n_b$: The block ID and num blocks.
Require: $t_{id}$, $n_t$: The thread ID and num threads.
Require: $\mathcal{L}$: Map cell $(i, j)$ to locked boolean (i.e., it is in $O \cup G$).
Require: $v$: The harmonic function values in log-space.
Require: $t$: The iteration index.
 1: for $i = b_{id}$; $i < m_1$; $i = i + n_b$ do
 2:     $k \leftarrow (t\%2) \neq (i\%2)$
 3:     for $j = t_{id} + k$; $j < m_2$; $j = j + 2n_t$ do
 4:         if $\mathcal{L}_{i,j}$ = false then
 5:             $v^* \leftarrow \max\{v_{i-1,j}, v_{i+1,j}, v_{i,j-1}, v_{i,j+1}\}$
 6:             $v_{i,j} \leftarrow v^* + \log(\exp\{v_{i-1,j} - v^*\} + \exp\{v_{i+1,j} - v^*\} +$
 7:                     $\exp\{v_{i,j-1} - v^*\} + \exp\{v_{i,j+1} - v^*\})$
 8:         end if
 9:     end for
10: end for
```

We use CUDA [18], a General-Purpose GPU (GPGPU) language developed by Nvidia. Recently, low-power embedded GPUs have become available for robotics applications (e.g., Nvidia Jetson TX1) which have hundreds of cores. In CUDA, *threads* are grouped into collections called *blocks*, both denoted by their index (zero-indexed). Threads in a block are executed together on a streaming multiprocessor. The actual unit that is executed in parallel on an Nvidia GPU is called a *warp*, consisting of 32 threads within a block (for current hardware). For this reason, blocks sizes are commonly divisible by 32; we use 1024 in the experiments. Algorithm 1 describes the parallel log-space Gauss-Seidel implementation (2-D) for threads, updating v at particular cells. It *strides* over multiple cells to handle scenarios in which the number of blocks and/or threads is less than the grid's height and/or width.

IV. Experimentation

We first compare non-log-space implementations with the log-space implementations, both CPU and GPU in 7 domains in Table I. The UMass [19] and Willow Garage [8] domains model path planning in a known office/home environment. The Mine domain captures path planning in maps produced by Simultaneous Localization and Mapping (SLAM) algorithms generated from point clouds [20]. The $\mathcal{C}$-Space domain is representative of basic motion planning for a two degree of freedom (DOF) manipulator [21]. The Maze domain describes often the worst-case scenario for path planners: narrow corridors and large sizes [22].

The results clearly show that CPU SOR ($\mathcal{C}$) fails to properly converge in the percentage of free space cells (skipping obstacles and goals) with a valid streamline (%). This is, of course, due to numerical precision issues. Both log-space CPU and GPU yield a 100% for this metric. The results also show the benefits of our GPU implementation (ℓ-$\mathcal{G}$) over our CPU implementation (ℓ-$\mathcal{C}$). The GPU version is one to two orders of magnitude faster than the CPU version. Since the update times (T_u) are low, we are able to easily run any number of updates and return a path anytime (albeit not immediately optimal). Additionally, the convergence times (T_c) are universally quite low for the log-space GPU implementation. For these reasons, the overall method is easily usable in robot scenarios that must move and react in real-time. Finally, we visually demonstrate the log-space mapping produces superior streamlines, as compared with non-log-space implementations (both single- and double-precisions), in Figures 1 and 2.

		$\mathcal{C}$			ℓ-$\mathcal{C}$		ℓ-$\mathcal{G}$	
Domain	N	%	T_u	T_c	T_u	T_c	T_u	T_c
UMass	152600	90.50	0.74	0.51	6.27	160.51	0.19	4.88
Willow	1694561	00.67	7.79	2.64	63.81	7095.86	0.39	42.55
Mine S	1221120	20.25	2.63	1.31	16.48	786.32	0.26	12.37
Mine L	2461965	10.24	3.71	0.99	18.57	887.45	0.36	16.96
C-Space	95352	35.67	0.37	0.09	2.94	67.90	0.08	1.86
Maze S	194084	01.71	1.03	0.09	9.21	185.12	0.08	1.62
Maze L	925444	00.14	5.00	0.37	44.19	8321.81	0.28	51.83

TABLE I

RESULTS FOR 7 DOMAINS WITH DIFFERING NUMBERS OF CELLS N. ALGORITHMS: CPU SOR ($\mathcal{C}$), CPU LOG-GS (ℓ-$\mathcal{C}$), GPU LOG-GS (ℓ-$\mathcal{G}$). METRICS: PERCENTAGE OF CELLS WITH A VALID STREAMLINE (%), TIME PER UPDATE (T_u IN *milliseconds*) AND TIME UNTIL CONVERGENCE (T_c IN *seconds*).

We demonstrate the new long-range, fast path planning capabilities that the algorithm provides in a real system. Figure 3 depicts an experiment using the log-space harmonic function ROS node on the uBot-6. The log-space solution enables the uBot-6 to plan a smooth path from one corner of the building's floor to the other. Since it is an anytime algorithm, the uBot-6 is able to begin executing a plan almost immediately with respect to real-world time. The solution improves as it follows the streamline, reaching the destination quickly while optimally avoiding obstacles.

V. Conclusion

We present a log-space solution to the long-standing numerical precision problem for harmonic function path planning. This approach leverages the log-sum-exp algorithm to do neighbor averaging to operate within numerical precision. We prove three propositions regarding the algorithm's correctness and convergence properties. Experiments on 7 realistic domains clearly show the proposed method allows for exponentially larger problems to be solved. We propose a GPU implementation as well, which provides over an order of magnitude speedup. We demonstrate the approach on a real robot as it rapidly produces an optimal obstacle-avoiding path to navigate a building environment. A new library called *epic* contains both CPU and GPU implementations, a *nav_core* plugin for ROS, and a more general anytime harmonic path planning ROS node. We provide all of the source code for solving harmonic functions in log-space so that other researchers may continue to design large-scale robotic path and motion planning algorithms.

References

[1] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 22, no. 2, pp. 224–241, 1992.

[2] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, *et al.*, "Stanley: The robot that won the DARPA Grand Challenge," *Journal of Field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

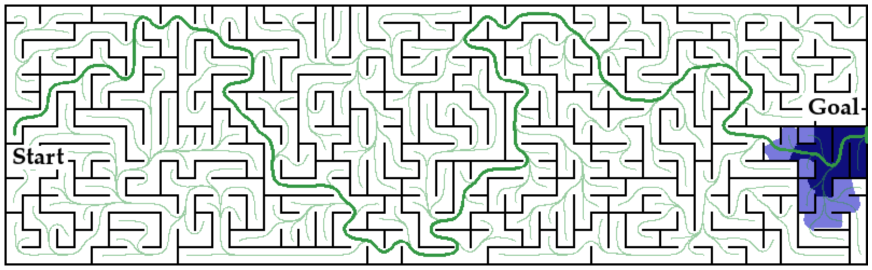

Fig. 2. Example Maze S [22] (802-by-242) with obstacles (black) and free space (white and blue). Example streamlines (green) show our log-space Gauss-Seidel GPU implementation's solution. *All* non-obstacle cells return a valid gradient for our algorithm. The two small blue regions are the only cells in which valid gradients (producing goal-reaching streamlines) are shown for *non*-log-space SOR with single-precision (dark blue) and double-precision (both blue). This figure also highlights the exponentially longer paths that the algorithm can find, as given by the highlighted streamline (Start to Goal).

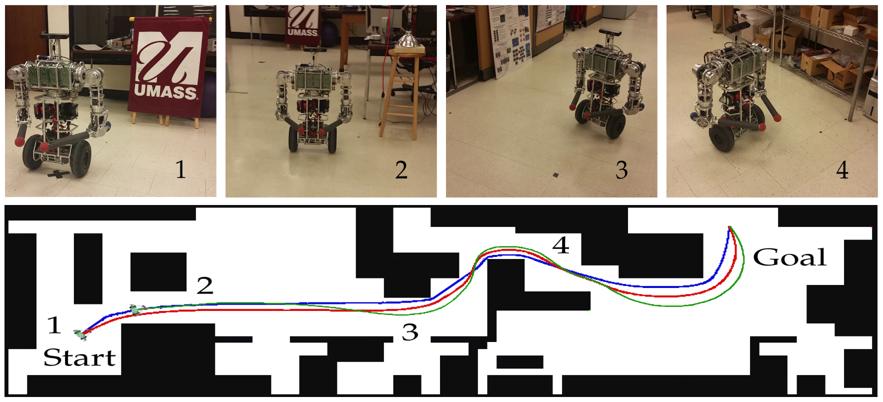

Fig. 3. Demonstration of uBot-6's log-space harmonic path planning made over time (top 1-4) in a laboratory map (940-by-310). The complete path and map are shown (bottom). Blue, red, and green lines show the paths after 0.5, 1.5, and 4.5 seconds of planning, respectively. Path following and execution started after receiving the first valid path at 0.5 seconds. During execution the path is refined until convergence. This demonstrates the anytime properties.

[3] R. Hong and G. N. DeSouza, "A real-time path planner for a smart wheelchair using harmonic potentials and a rubber band model," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2010, pp. 3282–3287.

[4] C. Goerzen, Z. Kong, and B. Mettler, "A survey of motion planning algorithms from the perspective of autonomous UAV guidance," *Journal of Intelligent and Robotic Systems*, vol. 57, no. 1-4, pp. 65–100, 2010.

[5] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2. IEEE, 2000, pp. 995–1001.

[6] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[7] C. I. Connolly and R. A. Grupen, "The applications of harmonic functions to robotics," *Journal of Robotic Systems*, vol. 10, no. 7, pp. 931–946, 1993.

[8] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, "The office marathon: Robust navigation in an indoor office environment," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2010, pp. 3–8.

[9] J. Rosell and P. Iniguez, "Path planning using harmonic functions and probabilistic cell decomposition," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2005, pp. 1803–1808.

[10] D. Aarno, D. Kragic, and H. Christensen, "Artificial potential biased probabilistic roadmap method," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, vol. 1. IEEE, 2004, pp. 461–466.

[11] S. Garrido, L. Moreno, D. Blanco, and F. M. Monar, "Robotic motion using harmonic functions and finite elements," *Journal of Intelligent and Robotic Systems*, vol. 59, no. 1, pp. 57–73, 2010.

[12] C. Torres-Huitzil, B. Girau, A. Boumaza, and B. Scherrer, "Embedded harmonic control for trajectory planning in large environments," in *Proceedings of the International Conference on ReConFigurable Computing and FPGAs*, 2008.

[13] S. Scherer, S. Singh, L. Chamberlain, and M. Elgersma, "Flying fast and low among obstacles: Methodology and experiments," *The International Journal of Robotics Research*, vol. 27, no. 5, pp. 549–574, 2008.

[14] "IEEE standard for floating-point arithmetic," *IEEE Std 754-2008*, pp. 1–70, August 2008.

[15] A. Tahbaz-Salehi and A. Jadbabaie, "A one-parameter family of distributed consensus algorithms with boundary: From shortest paths to mean hitting times," in *Proceedings of the 4th IEEE Conference on Decision and Control (CDC)*. IEEE, 2006, pp. 4664–4669.

[16] D. Ruiken, M. W. Lanighan, and R. A. Grupen, "Postural modes and control for dexterous mobile manipulation: the UMass uBot concept," in *Proceedings of the 13th IEEE-RAS International Conference on Humanoid Robots (Humanoids)*, October 2013, pp. 280–285.

[17] C. I. Connolly, "Harmonic functions and collision probabilities," *International Journal of Robotics Research*, vol. 16, no. 4, pp. 497–507, 1997.

[18] Nvidia Corporation, "About CUDA," 2016. [Online]. Available: https://developer.nvidia.com/about-cuda

[19] S. Sen and R. A. Grupen, "Integrating task level planning with stochastic control," School of Computer Science, University of Massachusetts Amherst, Tech. Rep. UM-CS-2014-005, 2014.

[20] S. Thrun, D. Hähnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker, "A system for volumetric robotic mapping of abandoned mines," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Taipei, Taiwan, 2003.

[21] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to autonomous mobile robots*. MIT press, 2011.

[22] J. Boström and JGB Service, "Maze generator," 2016. [Online]. Available: http://www.mazegenerator.net/