Spring 2021 ME/CS/ECE759 Final Project Report
University of Wisconsin-Madison

Comparing GPU-Accelerated Artificial Potential Fields and Harmonic Potential Field Mobile Path Planning Implementations

Mike Hagenow
Kevin Welsh

May 7, 2021

Abstract

Path planning is a ubiquitous problem in robotics which consists of finding a collision-free path between a specified start and goal position. This report explores two less common methods for low-dimensional planning spaces: Artificial Potential Fields (APF) and Harmonic Potential Fields. Artificial potential fields are capable of fast, real-time planning, but highly subject to local minima. Harmonic potential fields are theoretically free of local minima, but have other practical concerns and are more computationally expensive to compute. In this report, we provide implementations, considerations, and lessons learned from implementing both of these methods. A resulting comparison indicates that the methods represent a tradeoff between accuracy and computational speed. However, our results indicate that while these methods offer interesting theoretical bases, they are still likely inferior in practice to other common planning methods (e.g., RRT, A-star).

Mike primarily focused on Harmonic Potential Functions.
Kevin primarily focused on Artificial Potential Fields.
Mike and Kevin both worked on utilities.

Link to Final Project `git` repo:
https://euler.wacc.wisc.edu/mhagenow/me759-final-project-path-planning

Contents

1. General information

1. Your home department: Mechanical Engineering (Hagenow), Computer Science (Welsh)
2. Current status: PhD student
3. Individuals working on the Final Project:
    - Mike Hagenow
    - Kevin Welsh
4. Choose one of the following two statements (there should be only one statement here):
    - I release the ME759 Final Project code as open source and under a BSD3 license for unfettered use of it by any interested party.

2. Problem statement

Path planning is a ubiquitous problem in robotics which consists of finding a collision-free path between a specified start and goal position. There exist many methods for solving such problems, which often balance criteria such as computational complexity, accuracy, and optimality. When the planning space is high dimensional, the most commonly used methods are sampled-based (e.g. RRT [1]). As a result, fewer implementations and benchmarks are available for lower-dimensional potential field methods which can be valuable for mobile path planning. In this project, we explored two less-benchmarked approaches for mobile path planning: artificial potential fields and harmonic potential fields and performed a comparison between methods.

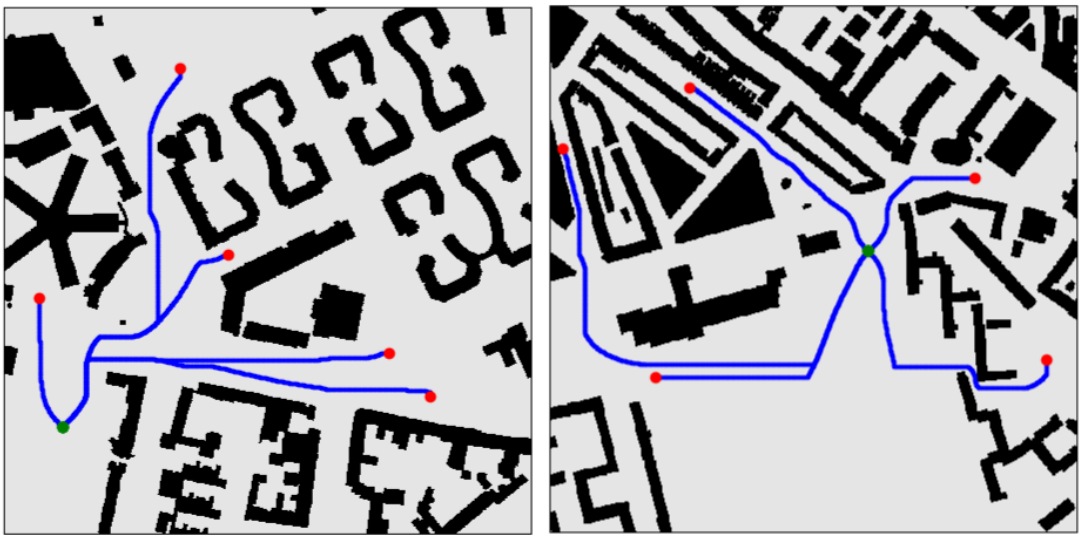

Figure 1: Examples of successful path planning from the algorithms implemented in this report from starting points (red) to a common goal point (green). Blue represents the paths and black represent collisions in the scene.

This project explores parallel implementations of two path planning approaches: artificial potential fields and harmonic potential functions. Artificial potential fields are a simple approach where paths can be calculated in real time, but methods often suffer from local minima in finding solutions. Harmonic potential fields are a more computationally expensive approach which are theoretically free of local minima, however, the large number of iterations and time required to get a reasonable solution is often

prohibitive. In this project, we describe various solutions and implementations for the two planning methods. This project compares the methods over a collection of 2-D urban maps. Each map is represented by a binary image that encodes the free and occupied space on the map. For each map, several start and end points were hand selected to exhibit a distribution of perceived difficulties with respect to local minima and path distance.

3. Solution description

a. Artificial Potential Fields

Artificial potential fields are a straightforward way to formulate the path planning problem. At a high level, the environment is modeled with regions of high potential energy, while the intended goal is modeled as a region of low potential energy. Thus, the problem is to find a path from the current position to the goal position while minimizing the potential energy along the path. The simplest way to do this is through a standard gradient descent algorithm, which greedily attempts to reduce the potential as much as possible at every step. This has the benefit of being a simple and efficient algorithm to implement. However, as is evident from the nonlinear optimization theory, any optimization method only has hope to find a local minimum and not the intended global minimum. These characteristics mean that this problem formulation is primarily conducive to scenarios where high performance control is necessary and the complexity of the environment is limited (i.e., short distance drone control). All of the methods described below use the following form for the goal potential, V_g, and the environment potential, V_e:

$$V_g = k_g||x - x_g||^2,\ V_e = k_e\left(\frac{1}{d+1} - \frac{1}{D+1}\right)^2 \qquad (1)$$

Where k_g and k_e are the goal and environmental gains, respectively, d is the distance to the environment, and D is the max distance to consider a potential collision.

Euclidean Distance Transform

The artificial potential field that is navigated consists of the sum of two pieces. The first is a function of the distance to the goal, while the second is a function of the distance to the environment. To compute the second, the euclidean distance transform of the binary environment image was first computed. To accomplish this, the algorithm described by Meijster, Roerdink and Hesselink [2] was implemented. This algorithm was chosen because it is faster than the naive implementations, and can be readily parallelized using OpenMP compiler directives.

The algorithm is broken down into two sections. The first section computes the 1-D distance transform for each of the columns of the image. This can be easily parallelized because each of the columns are treated independently. The second section computes the distance transform for each of the rows using the previous computation to determine which point is the closest. Each row is treated independently, so it can also be parallelized similarly. Since the second section requires more repetitive data reading, the 2-D image was stored and processed in row-major format to maximize data locality.

Bacterial Foraging

The primary drawback of naive artificial potential field methods is that it only finds local minima. However, in the path planning case, we are not trying to discover the global minimum, but instead find a path toward it. Bacterial foraging is one method that aims to increase robustness to local minima by searching multiple paths in a local region at once [3]. At each step, several subqueries are performed by

starting with a fixed number of nearby points and performing a low number of gradient descent steps on them. The trajectory that gets the closest to the goal then becomes part of the main trajectory, and the rest are discarded. By injecting this discrete random jump to a nearby point, the algorithm can hop out of local minima. These subqueries were implemented in parallel using OpenMP. Unfortunately, this local minima escape strategy also allows the algorithm to hop through thin walls and return infeasible trajectories. Figure 4 shows this algorithm navigating smoothly, but passing through a very thin wall.

Genetic Descent

Genetic gradient descent is motivated by the observation that some planning problems can be made solvable by gradient descent if certain parameters of the potential are optimized. Genetic descent uses a straightforward genetic optimization algorithm to modify the parameters k_g and k_e of the artificial potential to determine a set of parameters that get closest to the goal [4]. This was implemented in a three step process. Firstly, the current population of parameters is evaluated in parallel using a gradient descent algorithm. Then, the top performing sets of parameters are chosen to "reproduce" by averaging their parameter values together. Lastly, a small perturbation was added to represent "mutation" of the genes. This cycle is repeated for a fixed number of iterations or until the goal is reached. This reasonably effectively combines exploration of the parameter space (through mutation) and exploitation of successful parameters (through reproduction) to arrive at an optimal set of parameters. Figure 4 shows how a slight change in gain parameters can cause a previously failing gradient descent process to succeed.

Random Descent

Similarly to genetic descent, random descent is motivated by the observation that gradient descent may be able to solve similar problems to the one at hand. In this case, however, we consider achieving the same goal position from different starting positions. This is done in two phases. Firstly, we randomly sample free points on the map and attempt to plan a path to the goal from there. If we are successful, we add that trajectory to the set of known solutions. Then, we try to plan a path from our starting point to the goal or to the starting point of one of our known good trajectories. If we fail to achieve that, we repeat the cycle and attempt to expand our list of known good trajectories. This part of the planning process can be performed in parallel. This has the unique ability to solve problems that are otherwise impossible for standard gradient descent methods to solve, like having to temporarily move away from the goal. However, this also means that the solutions that it does generate may not be optimal or appear sensible. This limitation is very similar to the limitations of popular random sampling planning algorithms, like RRT. Figure 4 shows this alternative goal seeking behavior well, where two paths appear to be going the wrong way initially, and in this case only one successfully meets up with a known solution.

b. Harmonic Potential Fields

Harmonic potential fields are methods for path planning where the potential fields are iteratively solved to construct a local minima free potential from any starting point in the space to a set attractor point. In contrast to pure gradient methods and some sampling methods, this can be considered a multi-query planner. In other words, once the potential function has been solved, it is trivial to construct new paths from different starting points to the same attractor point. This can be useful in cases such as homing a mobile robot (i.e., if the robot is low on power, it wants to be able to return to its docking station from any point in a room).

The harmonic potential field approach consists of solving a function that is a solution to the Laplacian. The process is similar to heat transfer and fluids approaches to diffusion. We present a basic review of the approach for context[1]. Laplace's equation is imposed as a constraint on a discretized representation of the configuration space. The goal attractor point and collisions are imposed as further boundary condition constraints (e.g., Dirichlet, Neumann) on the solution. The Laplacian can be expressed as:

$$\nabla\phi = \sum_{i=1}^{n} \frac{\partial^2 \phi}{\partial^2 x_i^2} = 0 \quad (2)$$

For 2-dimensional planning spaces, the Laplacian can be enforced for a given point via the following discretization:

$$u^{n+1}(x_i, y_j) = \frac{1}{4}[u^n(x_{i+1}, y_j) + u^n(x_{i-1}, y_j) + u^n(x_i, y_{j+1}) + u^n(x_i, y_{j-1})] \quad (3)$$

where n is the previous iteration. Proper handling must also consider edge cases. After the Laplacian is calculated, a simple gradient descent can traverse from the starting point to the global minimum. In this project, solutions were explored ranging from simple global memory access in CUDA to tiling using shared memory and attempts to hasten convergence. The following sections detail experiments and iterations of the implementation which were assessed to determine the final implementation for comparison with the artificial potential field methods.

CUDA implementation details

In this project, all solutions revolve around 2D planning spaces which are solved by parallelizing calculations of each successive iteration of the grid. A python serial implementation is also provided for comparison of performance. To avoid in-place updates of the matrix during iterations, each CUDA kernel had an input and output matrix argument. Between iterations, the pointers for memory were flipped to allow for successive iterations. Additionally, to enforce boundary conditions, all kernels contained a constant mask matrix which enforced the saddle point and collision points.

Use of shared memory can be valuable when threads in a block have repeated access to the same location in memory. In Harmonic potential fields there is some, yet limited repeated memory access. To explore whether this repeated access could improve performance, we implemented a tiled shared memory approach using a 2D grid configuration. The tile size was maximized (i.e., 32x32) and the total shared memory size was padded by one on each side to allow for Laplacian calculations for the edges of the tile.

Choice of Data Type

One early issue with the approach was finding an appropriate data representation to assure there was sufficient data resolution to allow for subtle differences between discretized points. To allow for a local-minima free solution, the data needs to be able to monotonically decrease towards the saddle point. Depending on the size of the grid, this can require a large number of unique grid values. Early testing found that inappropriate data types, such as 32-bit integers and floats, were subject to round-off error which did not allow convergence for the paths, even for small problem sizes (e.g., 1073x1073).

[1] more thorough reviews are available online (e.g., http://robotics.stanford.edu/~mitul/rmp/)

To address this, we leveraged the inherent bounds of the grid points. When using Dirichlet boundary conditions, there is a minimum value for saddle points and a maximum value (repulsive field) for collisions. In the Laplacian discretization, all other points are averages of neighboring points and are therefore guaranteed to fall in the range between these values. Thus, our final solution utilized a 64-bit datatype (i.e., unsigned long long int) and set the saddle and collisions to zero and one fourth of the maximum value, respectively. The maximum was set to one fourth of the data type maximum to avoid overflow when accumulating the sum of neighbors. It would also be possible to divide each entry prior to summing, however, this could lead to additional truncation (i.e., precision loss). This approach was generally able to resolve the data type issue for our given problem size. For larger problem sizes, another remapping may be required such as logarithmic mapping [5].

Exit conditions

To determine the appropriate number of iterations of the CUDA kernel call, simple exit conditions were used. First, there was a maximum number of kernel iterations that could be executed. Second, a running count was kept during the kernel to count the number of entries that had changed over a certain threshold. This was accomplished using an atomic operation to avoid race conditions. A simple counter without the atomic operation was also implemented to see if there would be a performance improvement (since the precision of the count wasn't crucial), but it made negligible difference. During most runs, the threshold was set to when all entries changed by zero.

Boundary conditions

Boundation conditions are commonly enforced either using Dirichlet or Neumann boundary conditions. The Dirichlet boundary condition is enforced by constraining the value of the potential field directly:

$$\phi|_{i,j\in C} = c \quad (4)$$

The Neumann boundary condition is enforced by limiting the derivative of the potential field (i.e., disallowing the gradient flow through a collision):

$$\frac{\partial \phi}{\partial n}|_{i,j\in C} = 0 \quad (5)$$

Both methods were implemented for comparison. Neumann boundary conditions are often implemented by solving a linear system which enforces a discretized representation of the derivative. In this project, both the Dirichlet and Neumann boundary conditions were enforced iteratively on each point in the grid in order to maintain the structure within an iterative CUDA kernel. In the future, it would be interesting to compare a least-squares solution using a CUDA-optimized library. The Dirichlet boundary conditions were enforced by applying a mask of the constraints after each iteration (i.e., restoring to the proper values). The Neumann boundary conditions were enforced by applying a three-point derivative approximation to each collision in the mask. Since there were two partial derivative directions for the 2D space, the final value was the average of the values satisfying the derivative in each direction. While sometimes the boundary conditions are only enforced on the boundary of collisions, we applied the boundary to any collision point for simplicity. It would also be possible to calculate the concave hull of the grid prior to the iterations to limit the mask to boundaries.

In our testing, it was found that the Dirichlet boundary conditions were more simple and more easily converged to appropriate results. One potential advantage of Neumann boundary conditions is that solutions can more easily pass closely to collisions (as opposed to the repulsive fields of Dirichlet boundaries). However, in our testing, we were rarely able to get proper convergence and while segments of the solution were able to pass closer to obstacles, there were often errors where the path would also penetrate obstacles (see Figure 2).

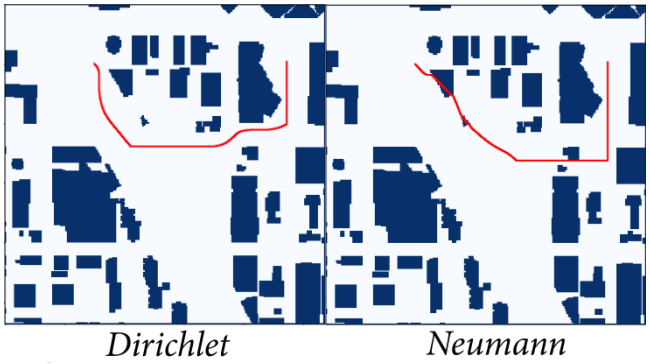

Figure 2: Comparison of boundary conditions. Both methods were run for 400,000 iterations. The Neumann boundary condition did not converge.

Successive Over Relaxation (SOR)

Iterative methods often employ techniques to try and increase the rate of convergence. One common technique is successive over relaxation (SOR) [6] which acts as a basic filter on updates to a particular cells's value:

$$\phi_{i,j}^{(k)} = \bar{\phi}_{i,j}^{(k)} + (1 - w)\phi_{i,j}^{(k-1)} \quad (6)$$

where w is an empirically chosen weighting factor which is tuned for the best convergence. While generally the value of w can range from zero to two with convergence, the data-type maximization discussed earlier does not allow values greater than one (i.e., these would lead to overflow). Thus, we limited testing to values between zero and one. As seen in Figure 3, we found that for our particular problem, the use of SOR did not improve the speed of convergence. This could be because there is a lack of overshoot in the iterations due to the averaging and thus, there is no need to add inertia and slow changes for the iterations.

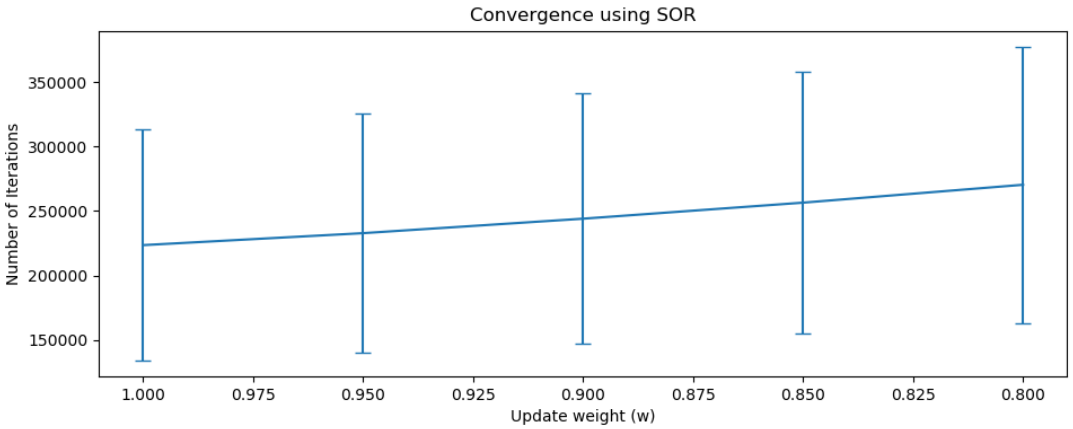

Figure 3: Effects of Successive Over Relaxation on convergence of the Harmonic potential function. Each weight was run for six examples. Number of iterations is defined as when all grid points settle to within 1e-6% of their final value (determined empirically). The results suggest SOR doesn't benefit this particular iterative algorithm.

Summary and Profiling

An implementation of Harmonic Potential functions was successfully implemented using CUDA. First, a basic python implementation tested the viability of the approach on a small problem size. This python implementation was extended to the full problem size with the intent of serving as a simple, readable baseline. This was compared to the CUDA global access and shared memory implementations to look at net performance gains as seen in Table 1.

Table 1: Kernel performance for each implementation

Method	**Python (Serial)**	**CUDA (Global Access - GTX 1050 Ti)**	**CUDA (Shared Memory and Tiling - GTX 1050 Ti)**
Iteration Time (ms)	8650	0.3953	0.3294

Additionally, the CUDA shared memory was profiled using Nvidia Nsight Compute[2] and a GTX 1080 for 1000 iterations of the kernel. The results confirm what was expected. The code is highly bound by memory-throughput. The SM utilization was only 22.19 percent whereas the memory utilization was 74.65 percent. The efforts for improving performance and lessons learned for the Harmonic functions are summarized below:

Techniques that did not yield benefits for this particular problem:

- Use of Neumann Boundary Conditions
- Increasing arithmetic intensity through repeated diffusion within a block between global synchronizations (did not converge)
- Successive over relaxation (SOR)
- Harmonic mean (did not converge and rounding error with ints)

Lessons Learned:

- For problems with minimal overlap of entries in computation, shared memory access gains can be minimal. Focusing on minimizing thread divergence and unnecessary computation and use of local variables can have just as large of an impact.
- Proper accuracy can be a large hurdle in scaling algorithms to more complex problems.

4. Overview of results and Demonstration of your project

We created a series of benchmark tests in order to assess all methods. We chose six candidate images from the Street Layout[3] image set. For each image, one goal position and five start positions were selected. The five start positions were selected by the authors in terms of increasing difficulty, noting that the most difficult would be challenging to solve using any of the proposed methods (e.g., many obstacles, long path). The results appear in Table 2.

[2] https://developer.nvidia.com/nsight-compute
[3] https://www.movingai.com/benchmarks/street/index.html

Table 2: Accuracy results for each method. Each entry contains three entries: number correct/ number incomplete/ number with collisions [e.g., through walls]). The harmonic potential fields had the highest accuracy, but took significantly longer to compute. Bacterial foraging was the most successful method using artificial potential fields.

	Gradient Descent	**Random Descent**	**Genetic Descent**	**Bacterial Foraging**	**Harmonic Fields**
Berlin	1/4/0	3/2/0	2/2/1	2/2/1	5/0/0
Boston	2/3/0	3/2/0	2/3/0	2/3/0	5/0/0
Denver	1/3/1	3/2/0	3/2/0	1/1/3	5/0/0
London	2/3/0	3/2/0	3/2/0	2/2/1	4/1/0
Milan	2/2/1	3/2/0	3/1/1	4/0/1	4/1/0
Moscow	1/4/0	2/3/0	2/3/0	0/3/2	3/2/0
Total	9/19/2	17/13/0	15/13/2	11/11/8	26/4/0

The Harmonic Potential method was able to identify solutions to the majority of point configurations. As seen in Figure 4 with the Milan scene, the greatest challenge came when the starting points were far from the goal and there were many collisions. This is likely due to a lack of representational power as the method converged relatively quickly. In other words, particularly when there are many nearby collisions, the averaging of the method runs out of unique values to represent the gradient without local minima. This could be addressed in the future by considering a logarithmic representation. However, the method was able to solve most other planning problems successfully.

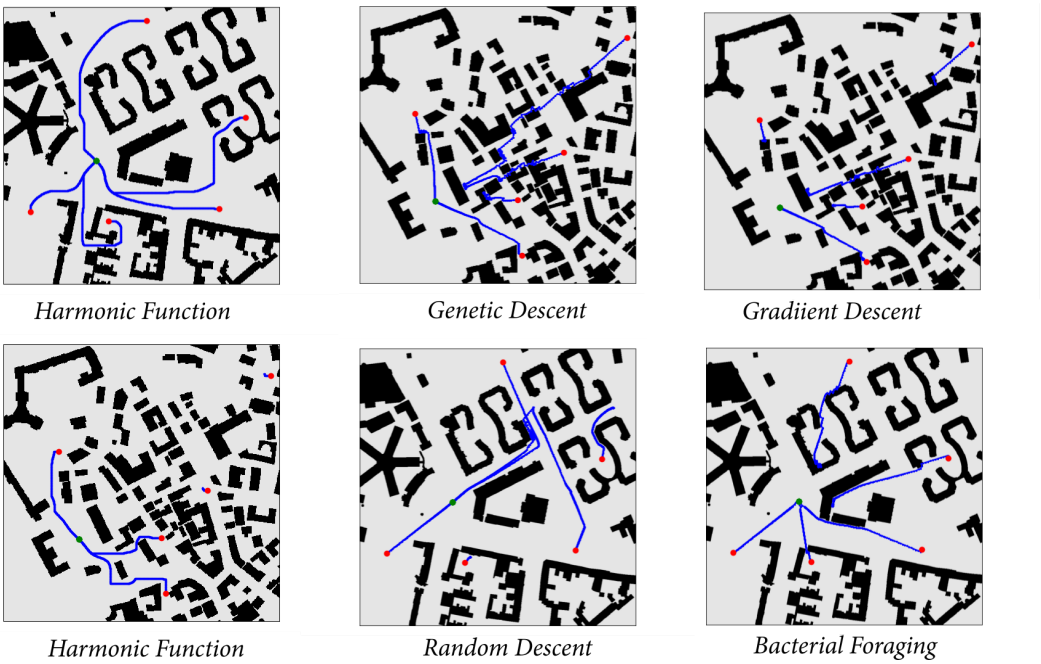

Figure 4: Example results from the benchmarking. (Top Left) Scene where harmonic potential fields were able to solve all paths. (Bottom Left) Harmonic potential fields were unable to diffuse to the top right due to numerous collisions which limited the ability to propagate the saddle point. (Top Middle) Genetic descent can complete a planning task that gradient

descent (Top Right) could not. (Bottom Middle) The random descent algorithm navigates away from the goal to eventually achieve it. (Bottom Right) The bacterial foraging algorithm passes through a thin wall on its way to the goal.

It is difficult to directly compare performance times between the methods as they accomplish different objectives. In Artificial Potential Fields, the goal is to determine a single path from start to end based on gradient steps. In our implementation, these methods took an average of 2 seconds to generate a solution or fail. The Harmonic Function method focuses its computational effort on creating a local minimum free solution that can be used on any query to the same goal position. Thus, the time is larger, but the solution can be used for multiple starting points. The Harmonic method is also generally more computationally expensive. Our implementation took an average of 333 seconds to compute one scene.

5. Deliverables

All code is available in the provided github including bash scripts to run various tests. The code is organized into folders: Harmonic, Artificial Potential Fields, Utilities, Scenes, and Benchmarks. More information is available in the github readme, but here we present an overview of the most important routines:

- *Scripts to launch routines*
  - compile.sh: compiles all C++, including CUDA and OpenMP
  - main.py: runs artificial potential fields and harmonic potential fields for all JSON benchmarks in the Benchmarks directory. Stores pngs of the results in the Results directory.
  - harmonic.sh: runs one example of the harmonic field and plots the results.
- *Benchmarks*
  - create_benchmarks.py: Uses a gui to select start and end points for all files in the Scenes directory. The benchmarks are then saved as JSON files.
  - show_benchmarks.py: Generates PNGs for the current benchmark files to visually confirm benchmark configurations.
- *Utilities*
  - sceneProcessor: converts pngs to CSV, plots harmonic potential fields, plots results and paths
- *Harmonic Functions*
  - harmonic_main: loads CSV, times kernel calls, and writes results to a processed CSV
  - harmonic_kernel.cu: contains the kernel launching, exit conditions, as well as a variety of kernels (e.g., global memory, neumann conditions, shared memory, looping/double buffered attempts)
  - pyHarmonic: serial python version mimicking the CUDA kernel (e.g., same datatypes, conditions)
  - testHarmonic: proof of concept that runs and plots a simple harmonic potential field solution (starting point for project)
- *Potential Fields*
  - main.cpp: loads CSV image, writes trajectories, implements several utility functions (joins trajectories based on proximities, computes gradients of a function, performs gradient descent on a function). Implements primary search algorithms: gradient descent, bacterial foraging, genetic descent, and random descent.
  - base_line.py: Serial python gradient descent visualization.

- DistanceField.cpp: parallel computation of euclidean distance transform from a binary image.

6. Conclusions and Future Work

Summarizing, the project was successful in implementing parallel versions of Artificial Potential Fields and Harmonic Functions that were able to solve some 2D planning problems. In the future, it will be interesting to continue to refine each of the methods. For Artificial Potential Fields, this would include combining and refining the tuning of the methods. For example, the gradient descent algorithm could be tuned to more quickly converge for specific potential functions, which would allow more exploration in all of the algorithms. For Harmonic Functions, this would include implementing methods to use logarithmic scales efficiently. Both approaches could also benefit from post processing of the produced paths. Once a feasible path has been achieved, optimization and relaxation methods could be applied to produce a shorter, smoother, or safer path. Lastly, given that the methods also have some failure points, it would be interesting to compare to methods that offer guaranteed completeness (e.g., A-star) and to compare performance.

References

[1] LaValle, Steven M. Planning algorithms. Cambridge university press, 2006.
[2] Meijster, Arnold, Jos BTM Roerdink, and Wim H. Hesselink. "A general algorithm for computing distance transforms in linear time." Mathematical Morphology and its applications to image and signal processing. Springer, Boston, MA, 2002. 331-340.
[3] Liu, Wei, et al. "Robot path planning using bacterial foraging algorithm." Journal of Computational and Theoretical Nanoscience 10.12 (2013): 2890-2896.
[4] Nagib, Gihan, and W. Gharieb. "Path planning for a mobile robot using genetic algorithms." IEEE Proceedings of Robotics 185189 (2004).
[5] Wray, Kyle Hollins, et al. "Log-space harmonic function path planning." 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2016.
[6] Hadjidimos, A. "Successive overrelaxation (SOR) and related methods." Journal of Computational and Applied Mathematics 123.1-2 (2000): 177-199.