Online Robot Navigation Using Continuously Updated Artificial Temperature Gradients

Yoav Golan, Shmil Edelman, Amir Shapiro, and Elon Rimon

***Abstract*—This paper suggests a novel method for a mobile robot to plan its path towards a target through an unknown environment, using sensors to discover new obstacles along the way. The method synthesizes an artificial temperature gradient throughout the known environment by numerically solving the heat conduction partial differential equation, where obstacles encountered during the robot navigation are "hot" and the target is "cold." The temperature at all other points on the known environment grid are computed numerically, and are continuously updated to account for new obstacles. This method ensures the creation of navigational potential field with no danger of being trapped in local minima. A computer simulation demonstrates the technique on several environment types. In addition, an experiment was conducted on a mobile robot that can navigate indoors and outdoors in real time using this method. The simulations and experiment show that the temperature gradient method is robust in its ability to find paths, and is practical for online sensor based navigation of mobile robots.**

***Index Terms*—Autonomous vehicle navigation, motion and path planning.**

I. Introduction

TODAY'S mobile robots are required to navigate in increasingly challenging dynamic and unknown environments such as industrial warehouses [1], [2], urban surroundings populated by pedestrians and vehicles [3], [4], and security duty while guarding sensitive installations [5]. This paper takes a close look at the classical potential field method, and suggests that numerical solutions can be computed on-line with extremely simple computation laws based on the heat conduction partial differential equation (PDE), with small computational latency time which makes such navigation systems practically viable.

Manuscript received September 14, 2016; accepted January 28, 2017. Date of publication February 7, 2017; date of current version March 10, 2017. This paper was recommended for publication by Associate Editor D. F. Wolf and Editor J. Wen upon evaluation of the reviewers' comments. This work was supported in part by the Helmsley Charitable Trust through the Agricultural, Biological and Cognitive Robotics Center of Ben-Gurion University.

Y. Golan, S. Edelman, and A. Shapiro are with the Department of Mechanical Engineering, Ben-Gurion University of the Negev, Beer-Sheva 8499000, Israel (e-mail: yoavgo@post.bgu.ac.il; shmulike@post.bgu.ac.il; ashapiro@bgu.ac.il).

E. Rimon is with the Department of Mechanical Engineering, Technion – Israel Institute of Technology, Haifa 3200003, Israel (e-mail: rimon@technion.ac.il).

This paper has supplementary downloadable material available at http://ieeexplore.ieee.org, provided by the authors. The Supplemental Material contains a video that showcases the simulations and experiments conducted in the paper. This material is 10.3 MB in size.

Color versions of one or more of the figures in this letter are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/LRA.2017.2665682

Path-planning using artificial potential fields [6] offers an elegant solution for choosing a navigational route by constructing an artificial potential function that "attracts" the robot towards the target and "repels" it away from the obstacles. However, it is exceedingly difficult to construct a potential field without residual local minima, as dealt with by [7], that may inadvertently cause the robot (which moves along the negative gradient of the field) to get stuck in the wrong place. This problem becomes even more difficult when information on the environment is collected on-line by the robot's sensors. Other well-known issues documented by Koren [8] include oscillations near obstacle boundaries, avoidance of entering very narrow passages, and oscillations when traveling through narrow passages.

This paper is concerned with safely navigating a mobile robot to a target while simultaneously acquiring new information about the environment. The robot's path is found with a novel potential field method that prevents "trap situations" [9], associated with the standard potential field methods mentioned above. In this paper, the mobile robot moves in a planar unknown environment, with a laser range sensor (LIDAR) such as the case in [10]. The robot also has a position sensor to determine its location in the environment. SLAM techniques [11] can be used for position estimation for a robot of this type, and is used in our experiment detailed in Section VI.

This paper demonstrates the application of sensor-based navigation analogous to the steady-state heat conduction solution. The temperature gradient navigation method eliminates problems that appear in potential field navigation and previous works on temperature gradient navigation discussed below.

The temperature gradient method substitutes potential with temperature, as we treat obstacles as *"hot"* or repelling, and the target as *"cold"* or attracting. The paper's main contribution is that the robot can use the heat conduction equation to numerically determine the solution on-line across the known environment, utilizing basic properties of heat transfer to solve well known weaknesses in the classical potential field method. As heat transferred by conduction travels from hot to cold, the resulting temperature gradient provides paths from any start point to the target by simply following the temperature negative gradient paths. The temperature gradient algorithm can also be applied to other types of motion planning problems, such as a robotic arm performing complex tasks in congested environments [12], [13].

Many robot navigation papers are concerned with the question *"how fast is this algorithm?"*. Our paper is concerned with a different question: *"is this algorithm fast enough?"*. Another

way to phrase the latter is: will the navigating robot need to physically pause or slow down its movement in order to allow its navigation algorithm time to adjust its path? We claim that the speed of an algorithm is irrelevant as long as its latent computation time is sufficiently short as to allow smooth navigation. Because the former question is usually asked, many modern algorithms may be extremely fast in certain environments, while they can be very slow in others, causing the flow of on-line navigation to be disrupted. This paper seeks to use a numerical solution that may be slower than some existing navigation methods in certain environments, but is *fast enough for any environment*.

II. Previous Work

Connolly [14] pioneered the idea of numerically solving the Laplace equation for navigational use in 1990. He demonstrated the ability of numeric solutions to find paths in two and three dimensional c-spaces for some simple, static environments. His method was implemented with the technology available at the time, and therefore lead to time frames that are unacceptable for on-line navigation (computation of simple environments took between 23 and 188 seconds!). He predicted that sufficiently powerful computers could make path construction using this method a viable option.

Sasaki [15] attempted in 1998 to synthesize a potential field that inherently has no local minima by using the elliptic PDE to solve heat conduction problems. By treating the start point as a hot point, the target as a cold one, and the obstacles as unknown adiabatic entities, he was able to construct a potential field that guarantees a path from the start to the target (if one exists) without local minima. However, Sasaki's method is only suitable for static and fully known environments. The start point was assigned a high temperature, and became a local maximum. Consequentially, paths could not be constructed that backtrack to the start point. This is important to us, as backtracking is an essential part of on-line navigation. As the work was done in 1998, computational limitations still prevented, or at least discouraged, application to higher dimensional problems or on-line navigation.

According to [15], computing the temperature gradient for a 31×31 grid consisting of 200 point obstacles, much more complicated than that in [14], took around 30 seconds! Under such a constraint, on-line robotic navigation is impractical. Using modern computers, we can now shorten the computation times enough for practical navigation. In our simulations, the time required to compute temperature gradients was shorter by a factor of over 10,000.

Other, more modern approaches for navigation using physical analogies have arisen, such as FM^2 [16]. FM^2 relies on the physical solution to *wave propagation* from one or more points. While the wave propagation method has many advantages, ours exhibits qualities that cannot be obtained by this method, such as true on-line navigation (FM^2 must re-start the algorithm when in the presence of newly discovered obstacles), an estimation metric for the quality of the navigation method (as discussed later), ease of implementation, and a variety of configuration options (such as a varying of local heat conduction coefficients to discourage paths in certain areas or directions).

A detailed comparison of the method with a creditable alternative method (RRT) highlights the contribution of this paper. The comparison is presented in the simulation section.

III. Artificial Temperature Gradient Navigation

This section introduces the heat conduction PDE and details the method in which we numerically synthesize the temperature gradient navigation field. We show that the temperature gradient synthesized in this way has no local minima. We focus on the synthesis of temperature gradient fields on static, fully known environments and expand the method to the more complex on-line problem in the next section.

A. The Heat Transfer by Conduction Equation

Let $T(x, y, t)$ denote the temperature at a point (x, y) at time t. In order to compute the temperature navigation field, we use the elliptic PDE describing two-dimensional heat transfer by conduction [17]:[1]

$$\frac{\partial}{\partial x}\left(k(x,y)\frac{\partial T(x,y,t)}{\partial x}\right)+\frac{\partial}{\partial y}\left(k(x,y)\frac{\partial T(x,y,t)}{\partial y}\right) = \frac{\partial T(x,y,t)}{\partial t} \tag{1}$$

Where $k(x, y)$ denotes the thermal conductivity coefficient, a system dependent non-negative parameter. In order to solve the on-line navigation problem using the heat transfer equation, we assume: 1) The thermal conductivity coefficient, k, is independent of location (x, y). 2) We focus on the steady state solution of (1), satisfying $\partial T(x, y, t)/\partial t = 0$. Under these assumptions, (1) simplifies to:

$$\frac{\partial^2 T(x,y)}{\partial x^2}+\frac{\partial^2 T(x,y)}{\partial y^2}=0 \tag{2}$$

B. Numerical Solution of the Heat Conduction Equation

The environment is discretized as a rectangular grid, where $N = n_1 \times n_2$ is the total number of cells. While we mostly discuss two dimensional problems, N can represent the number of cells in any number of dimensions with the same rules applied to a grid with $N = n_1 \times n_2 \times ... \times n_m$ cells. Each cell represents a physical space, and holds a unique temperature value.

By using finite difference approximation [18], one can numerically compute the temperature gradient using (2). The resulting finite difference equation is:

$$\frac{T_{i+1,j}-2T_{i,j}+T_{i-1,j}}{(\Delta x)^2}+\frac{T_{i,j+1}-2T_{i,j}+T_{i,j-1}}{(\Delta y)^2}=0 \tag{3}$$

Where i, j are indices of the temperature grid. By assuming square grid cells, $\Delta x = \Delta y$, one obtains a very simple solution for the temperature at the (i, j) cell, termed the Jacobi method [18]:

$$T_{i,j}=\frac{T_{i+1,j}+T_{i-1,j}+T_{i,j+1}+T_{i,j-1}}{4} \tag{4}$$

[1]There are other physical heat transfer models, such as convection, that seem less relevant for mobile robot navigation.

A similar, but faster method is the Gauss-Seidel method [18]. The method uses temperature points from the current iteration mid-computation, and yields better rates of convergence:

$$T_{i,j}^{k} = \frac{T_{i+1,j}^{k-1} + T_{i-1,j}^{k} + T_{i,j+1}^{k-1} + T_{i,j-1}^{k}}{4} \tag{5}$$

where the superscript k signifies the process iteration. Faster methods of computation exist (such as "succesive over relaxation" [18] and "immersed boundaries" [19]). However these methods are not discussed in this paper. The introduction of more advanced numerical method for this type of problem, such as those suggested by Saudi [20], [21], can greatly increase the applicability of the suggested navigation method to real situations.

A *single iteration* consists of the application of (5) on every cell in the temperature gradient in $\mathcal{O}(N)$ steps. An in depth analysis of the Laplace equation in robot navigation can be found in the work of Garrido [22]. One of the issues raised by Garrido is that Laplace equations with Dirichlet's boundary conditions (as is our case) may suffer from small numeric differences for away from the target, leading to rounding errors. We deal with this by using sufficiently high precision data types (up to 64 bits) for the temperature data storage, and by setting the *hot* and *cold* temperatures to the extreme values of the data type.

C. Proof of Absence of Local Minima

For the temperature gradient to be a suitable robot navigation tool, it should not contain local minima. By determining the target as a constant *cold* location, and the obstacles as constantly *hot*, we can show that given an initial guess of *hot* for all cells in the environment except the target, no local minima appear.

Lemma 1: A temperature gradient formulated with **hot** obstacles, a single **cold** target, and free space initialized as **hot** has a single minimum at the target.

Proof. Assume there is a local minimum with temperature C that appeared in the latest iteration. All cells except the target were initialized as **hot**. Therefore C has been reduced to its current value from **hot**. Based on (5), a cell is affected only by its neighbors. Hence the change in its temperature value was caused by one of its neighbors. Since the value of C is determined by the average of its neighbors, only a neighbor that has a value *lower* than C could lower the temperature from **hot** to C. If a neighboring cell has a value lower than C, the current cell cannot be a local minimum of the temperature field. ■

IV. On-Line Navigation by a Continuously Updated Temperature Gradient

This section describes our method of adapting the heat conduction solution to on-line navigation of a mobile robot with a LIDAR type sensor in two-dimensional c-space.

A. On-Line Navigation by Continuous Temperature Updates

A naive approach to on-line navigation would be to recompute the temperature gradient from an initialized state (all cells *hot*) every time the robot discovers new obstacle cells. This approach is extremely wasteful when one considers that adding a few new obstacle cells to an existing temperature gradient generally introduces only a *small* overall change in the gradient. This is true for obstacle cells that are part of a previously undiscovered obstacle, and even more so for additional obstacle cells on previously discovered obstacles.

In order to advantageously use the computations made up until the on-line discovery of new obstacle cells, we propose to *continuously update* the temperature gradient, adding obstacle cells when discovered and improving the solution whenever possible. We do this by maintaining a single temperature gradient that is constantly updated using (5). When a new obstacle cell is discovered in the environment, the temperature of this cell is set to *hot*, and the cell is added to the known obstacle list O (and as such remains at constant temperature). This creates a *disturbance* in the otherwise smooth gradient. This disturbance is resolved quickly, and the entire gradient field typically converges to the new steady-state solution within 0.1%–0.5% of the time required to re-compute the gradient from an initialized state according to the naive approach! A pseudo-code description of the on line algorithm follows.

Algorithm 1 On-Line Navigation.

```
1:  procedure NAVIGATION
2:    Target ← location of Target
3:    Start ← location of the robot's position
4:    ε ← convergence criterion
5:    O ← list of known obstacle cells
6:    counter ← 0
7:    every Cell ← Hot
8:    Cell(Target) ← Cold
9:  Initialization:
10:   while ε > max_dif(map) do
11:     for every cell not in O or Target do
12:       Cell_{i,j} ← (Cell_{i+1,j}+Cell_{i-1,j}+Cell_{i,j+1}+Cell_{i,j-1})/4
13:     if N iterations occurred and Cell(Start) = Hot then
14:     terminate (no path exists);
15: loop:
16:   O ← O+ new detected obstacle cells
17:   Location ← robot location
18:   every CellinO ← Hot
19:   for every cell not in O or Target do
20:     Cell_{i,j} ← (Cell_{i+1,j}+Cell_{i-1,j}+Cell_{i,j+1}+Cell_{i,j-1})/4
21:   Next_Step ← min(neighboring Cells).
22:   Drive_Robot_Towards(Next_Step)
23:   if robot is at a local minimum then
24:     Count Iterations in Location
25:     if count = N and FloodFill(Location, Target)
          = False then
26:     terminate (no path exists);
27:   if Location = Target then
28:     terminate (success);
```

The function max_dif in line 10 finds the largest change made in the temperature gradient from the previous iteration using a standard formula for determining gradient

convergence [18]:

$$max\_dif = \max_{0 \le i \cdot j \le N} \left(\frac{|T_{i,j}^{k-1} - T_{i,j}^{k}|}{T_{i,j}^{k}} \right) \tag{6}$$

The convergence criterion ε specified in line 10 is a user specified value. The smaller ε is, the closer the gradient is to the true solution. The function $FloodFill$ in line 25 is a simple $\mathcal{O}(N)$ test that the robot's current location and the target are in the same connected free-space, where N is the total number of cells in the environment.

B. Temporary Local Minima

The caveat with continuous update of the temperature gradient when new obstacle cells are discovered is that temporary local minima may appear. However, after several iterations these local minima disappear, since the heat conduction problem has a unique steady-state solution with a single minimum, regardless of the initial guess (Lemma 2). A major challenge is to obtain a worst-case upper bound on the number of iterations until a given temporary local minimum disappears. However, as discussed in the simulations section, in practice the number of iterations required for its disappearance is extremely small. Exceedingly low values of M may result in the robot reaching a local minimum, but this is a temporary issue that will self-correct as further iterations disperse the local minima.

Lemma 2: Local minima in a temperature gradient field disappear after a finite number of iterations.

Proof. The temperature field as we have described it is a Cauchy problem that satisfies the conditions of the Cauchy-Kowalevski theorem, and therefore has a unique solution. It is well known that the Gauss-Seidel method is guaranteed to converge to the solution for the heat conduction equation [18]. Lemma 1 shows us that certain initial conditions result in a solution with no local minima. If the solution is unique, arbitrary initial conditions result in the same temperature field with no local minima. ■

C. A Metric for On-Line Navigation Quality

It is important to quantify the quality of adding new obstacle cells to an existing temperature gradient rather than computing a new one. We know that a temperature gradient field with newly added obstacle cells will eventually provide the globally convergent navigational properties we desire. However, it is unacceptable that the robot need to pause its motion for the gradient to converge. The only alternative for the robot is to continue moving based on an un-converged temperature gradient field. For this reason, we introduce a metric M that quantifies the competence of a mobile robot using the on-line method in a given environment. The metric is a predictor of the viability of using the proposed navigation method.

The metric M measures the ratio between the *frequency* of new obstacle cells discovered on-line, and the robot's on-board processing power. The frequency of new obstacle cells discovery depends on the unknown layout of obstacle cells in the environment. However, it is reasonable to assume a correlation between the robot's speed and frequency of new obstacle cell discovery (a faster robot discovers more obstacles cells per second than a slower robot). The metric M is thus defined as the number of iterations performed on a temperature gradient field per unit distance the robot travels. This metric depends on the robot's average speed, ν, the environment size, N, the physical length of each cell, d, and the time required to compute a single-cell iteration of the heat conduction equation, P. The time required for a single iteration, denoted t_1, is the time required to compute all free-space cells, hence $t_1 \le N \times P$. Using these parameters. the metric M is given by the number of temperature gradient iterations performed each time the robot travels the length of a single cell:

$$M = \frac{t_1 \cdot d}{\nu} \tag{7}$$

The units of M are thus *number of iterations per cells traveled*.

An example of the computation of M for a real-world problem appears in Section VI. The simulation section shows that higher values of M lead to high quality paths that are closer to those generated by the naive approach, while lower values of M lead to lower quality paths. Since all the parameters that define M can be determined for any navigation task, the promise of continuous updates can be evaluated before the robot starts its navigation task. To create a reference *value* of M, one can compare the path length L of an on-line trial using the naive approach to a trial using continuous updates, given by:

$$Q = \frac{L_{\text{naive}}}{L_{\text{on-line}}} \tag{8}$$

One can expect that Q is smaller than unity, as the naive approach is extremely conservative. Hence the metric M is "good" when Q is close to unity, meaning that the path length generated using continuous updates is very close to the naive computation of the entire environment.

V. Simulation Results

In order to validate the on-line temperature gradient method, simulations were conducted in three different environments: maze environments, randomly generated environments, and "office floor" environments. A simulated robot that uses a 360° LIDAR sensor with a scanning radius of 5 length units. The robot's velocity ν is a simulation parameter, as is the number of iterations that can be performed per unit of time t_1.

1) Effects of Metric M: We started by evaluating the metric M by generating a series of 50 different 50 × 50 environment grids with randomized obstacles (binomial distribution, $p = 0.2$). Each environment grid was tested for a robot moving between opposite corners of the environment. In each environment a path was constructed from start to target (if possible), and the path length was measured. This was done for the on-line naive case (every step the robot takes, the entire temperature gradient is re-computed from a *hot* state), and using the continuous update method with several different robot speeds.

The average path length for each case is presented in Fig. 1. This graph shows that the path length of the method improves with higher values of M. Moreover, the quality Q approaches 1 at values of $M \cong 20$ for this type of robot and environment

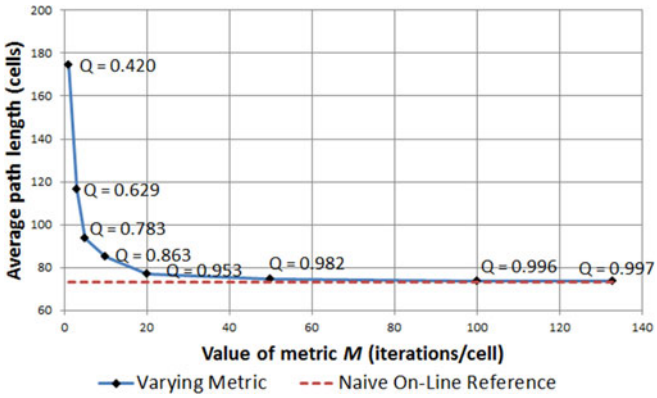

Fig. 1. Path lengths for different values of M. The red line (dashed) represents the average path length in the naive on-line method.

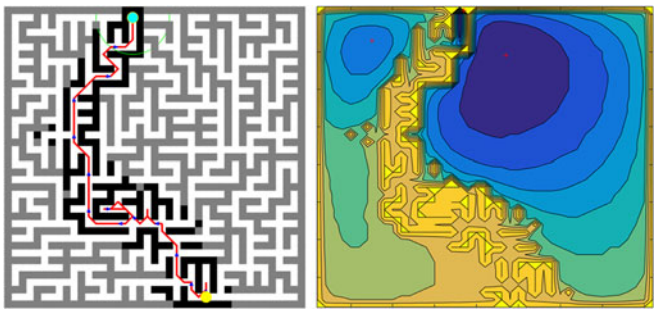

Fig. 2. On-line solution of a 40 × 40 maze (left). The start point is the yellow circle on the bottom. The target is the teal circle on the top. The red line is the path taken. Black squares are discovered obstacle cells, grey squares are undiscovered obstacle cells. The corresponding temperature map (right) is presented (with visual enhancement). Blue represents lower temperatures, yellow represents higher temperatures. Note that the appearance of local minima (marked with red crosses) implies that the temperature field is not yet converged. $M = 2$.

(higher values of M showed similar results). This means that a reasonable number of updates are equitable to the naive on-line navigation approach.

Based on the simulation, we wish to understand if the method is robust enough for real mobile robots. In each type of environment we tested the navigation method using several values of M. Simulation run-time was measured in each case to verify the validity of the robot-speed assumptions (if the on-line computation takes more time to run than the path length/robot speed, the robot would have to slow down). The method successfully navigated all environment types with all values of M. Low values of M consistently resulted in longer paths and longer-dwelling local minima. Fig. 3 shows the different paths taken using different values of M. Fig. 3(left) shows a path with a very low metric ($M = 3$), the path generated is longer, and has more backtracking. Fig. 3(right) shows a path with a moderate metric ($M = 25$), the path generated is more efficient (less backtracking) and its length is close to the naive approach path length.

In all cases the simulation run-time proved that the robot-speed assumptions were valid. However, it should be noted that in the case of a randomized environment with very low values of

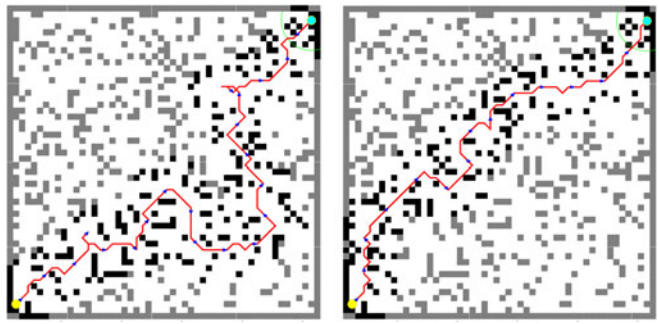

Fig. 3. Paths generated for different values of M in a randomized environment. The start point is the bottom left corner and the target is the top right corner. The left path was derived using $M = 3$, the right path was derived using $M = 25$.

M (≈1), sometimes the robot would arrive at a local minimum before it disappeared (due to the low update rate), and would dwell there for a number of iterations until the issue resolved itself (the local minimum disappeared). Even for low values of M we have noticed that the robot keeps a reasonable distance from obstacles, as a newly discovered obstacle cell has a great effect on its immediate surroundings within several gradient iterations.

A simulation of off-line three-dimensional c-space navigation was also implemented to demonstrate the applicability of the method to higher dimensions, however at this point on-line navigation was not implemented.

Comparison with other navigation methods: Within the scope of this paper, we compare our method only to one of the leading classes of navigation methods. One of the more auspicious algorithms is the Rapidly Exploring Random Tree (RRT) algorithm [23], [24], and variations of it (RRT* [25], RRT-Connect [26]). These algorithms are probabilistic and can be very fast in finding solutions to navigation problems. However, these algorithms have weaknesses that our method overcomes. First, RRT and its variations have difficulty navigating in heavily cluttered environments, as many of the random vectors generated cross obstacles and are eliminated, slowing the solution. Second, while RRT algorithms are able to pass through narrow passages, this generally takes many "guesses", as the probability of a guess vector pointing through the passage is low, making environments with multiple narrow passages or narrow corridors difficult to navigate. In both of these instances our method excels, as the method is indifferent to obstacle complexity and even benefits from it.

To better understand what degree of obstacle congestion renders RRT algorithms slower than ours, we implemented a version of RRT with dynamic restarts as described by [27]. The RRT parameters used were a goal bias of 0.9, a step size of 0.5, and a 20k iteration limit. We randomly generated 120 environments, with varying degrees of obstacle clutter. Both the RRT and the Temperature Gradient method ($M = 10$) were tested in each environment, and the time to reach the goal was measured. Fig. 5 shows the average computation times for each algorithm with different levels of obstacle congestion. It is clear that at lower levels of clutter, RRT is faster than our method.

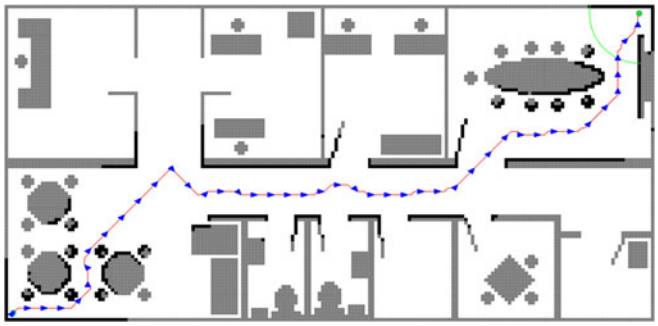

Fig. 4. A path planned in an $N = 195 \times 95$ "office floor" type environment from the bottom left corner to the top right, $M = 10$. The LIDAR sensor range is set to a 15 cell radius. Discovered obstacle cells are marked in black, undiscovered in grey.

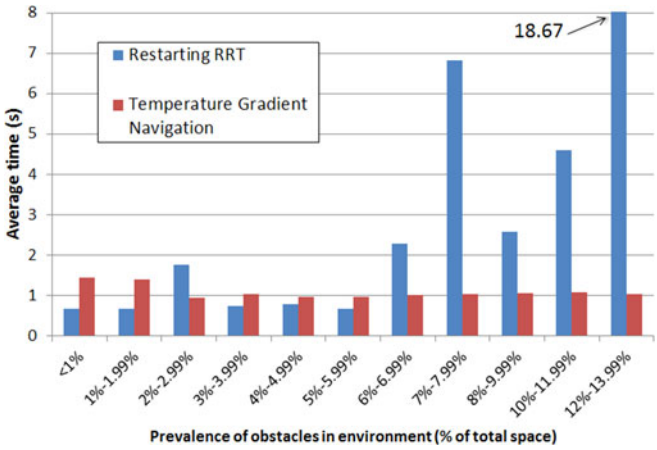

Fig. 5. Average computation time for differently congested environments using the suggested method and RRT with dynamic restarts. The horizontal axis represents the percentage of the environment that is an obstacle.

However, as we look at more cluttered environments the computation time of RRT begins to increase exponentially, while the proposed method decreases in computation time. The variance in the computation times in congested environments is also very large for RRT algorithms due to their inherent randomness. It is important to remember that we are concerned with whether or not the algorithm is fast enough, and not how fast it is. Therefore, as long as the algorithm permits the desired robot speed, there is no advantage in better computation times. One may observe Fig. 5 and realize that when using RRT, low levels of congestion permit ludicrously fast robots, while more congested environments only allow very slow robots. When using the temperature gradient method, the permitted robot speed is practically independent on the environment congestion (and is even increased with high levels of clutter).

We have also compared the run-time of our algorithm with off-line RRT* and Bi-Directional RRT (RRT-Connect) [28] in different environments using the RRT toolbox created by [29] and *RRT Explorer 3* [30]. In simple environments such as a room with several walls, RRT* found paths in much less time than our algorithm. However, in more congested environments, such as the maze in Fig. 2, RRT* required a large number of iterations to find a single solution (different biases and step sizes were tried). The time required to find a solution in the off-line maze using RRT* was in the order of 20 seconds (for the best case of parameters), Bi-Directional RRT did much better with 3.5 seconds on average. Still, this is the off-line solution. Our algorithm could find the *on-line* solution in *merely two seconds!* It should be noted that while often on-line navigation using RRTs includes restarting the algorithm, this is not always the case. Some alternative methods such as suggested by [27] utilize existing information by "chopping" branches that collide with newly discovered obstacles, and regrow the tree until a new solution is found. This method was implemented to achieve the results in Fig. 5.

VI. Experimental Results

In order to test the navigation method's applicability to real world problems, we conducted experiments with a mobile robot navigating inside our robotics laboratory and outdoors. Experiment Setup: For the experiments, we used a four-wheeled, two-motor driven mobile robot shown in Fig. 6(a). The robot uses a LIDAR sensor (SICK TIM551-2050001 detailed in [31]) with a range of 8 meters, an aperture angle of 270°, an angular resolution of 1°, and a scanning frequency of 15 Hz. The computer on-board the robot is an Intel NUC Mini PC using an Intel i5-6260U processor with 4 × 1.80 GHz cores. The robot runs the Robot Operating System (ROS). The robot uses SLAM to determine its location, the SLAM algorithm is implemented in ROS and is based on Monte-Carlo Localization (MCL) [32]. The robot's average speed is $\nu = 0.5$ meters per second (limited by safety considerations). The environment was discretized to square 0.1 × 0.1 m cells. The number of cells changed dynamically by using a distance-to-target heuristic (that is not elaborated in this paper), with typical values of $N \leq 76,000$. By combining the SLAM algorithm and our navigation algorithm, the robot is "aware" of: the location of the target, the robot's location, orientation and speed, the location of discovered obstacles (inflated by the robot's radius), and the desired path to the target. With this knowledge the robot can apply a simple control loop to follow the path until the target is reached.

A series of 10 navigation experiments were performed on the robot. Each experiment was conducted at least 3 times to test consistency, for a total of 30 trials. Between the robot and the target were non-staged obstacles- walls, furniture etc. that are part of the Ben-Gurion University Robotics Lab. The target was typically placed between 10 and 70 meters from the starting positions (path-wise, not aerial distance). Experiments included navigation from one room to another, outdoors-to-indoors, indoors-to-outdoors, and impossible tasks (no solution is possible). In all the experiments the robot reached the target (when possible). Moreover, the robot did not stop during the trials, and did not collide with any detected obstacles. The average distance traveled by the robot in each trial (excluding impossible missions) was 52.5 meters.

The robot performed one iteration of the temperature gradient every 0.0036 seconds on average, or 278 iterations per second. This provided the metric value of $M = 278 \cdot 0.1/0.5 =$

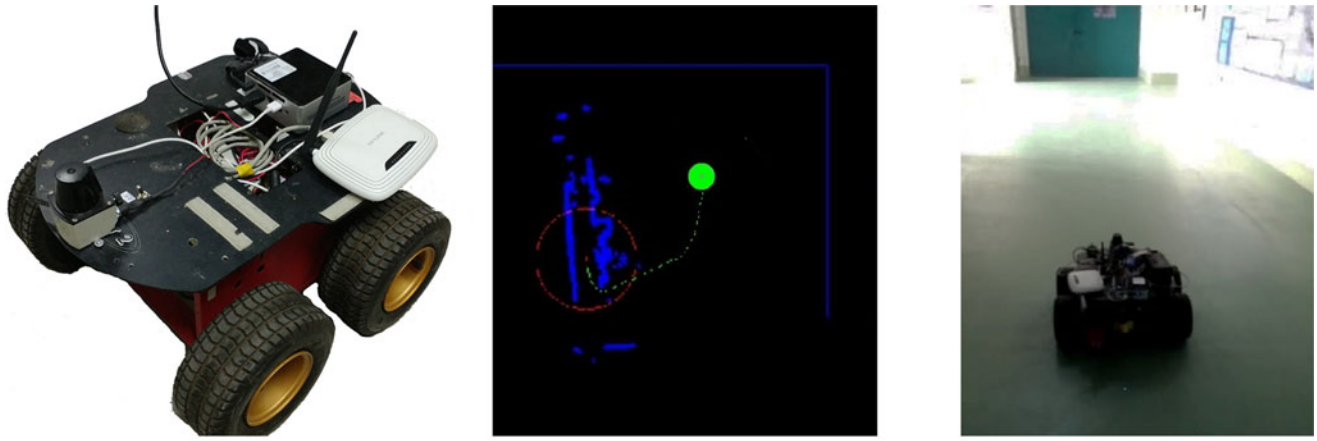

Fig. 6. Depiction of an experiment. (a) The mobile robot equipped with a LIDAR sensor and an on-board computer.(b) The robot's ROS interface. The robot's location and LIDAR scan radius are in red, obstacles in blue, the path to the target as a dotted green line, and the target is the center of the green circle. (c) The robot driving through a hallway trying to reach a goal outside.

55.6 $iterations/cells\ travelled$. An example of a trial environment is depicted in Fig. 6(c), and the ROS interface in the same trial is depicted in Fig. 6(b).

VII. Conclusion

This section discusses the navigation method, especially in light of the simulation and experiment results. The method is then compared to other modern navigation methods. The on-line algorithm has the following attributes.

First, the navigation method is *complete*. When the temperature map is initialized, such as in the beginning of on-line navigation, the "Start" cell is guaranteed to attain a temperature lower than the *hot* temperature within N iterations if the goal can be reached. This is because every iteration affects at least one more cell that is part of a viable path to the target. If after N iterations the "Start" temperature is *hot*, there is no path to the target. In on-line navigation the situation is not as simple. Mid-navigation, the robot could discover obstacles that prevent a viable path to the goal. This would lead the robot to a local minimum with no indication of termination. Luckily, we can use the knowledge that local minima disappear quickly to resolve this issue. We determine a fixed amount of iterations the robot can stay in a local minimum (an indication of the possibility that there is no path) before performing a "Flood-Fill" test. This addition guarantees the program terminates if no path is possible.

Second, the navigation method is not slower in more congested environments (more obstacle cells in a given area). This is because every iteration entails the same order of computations, at most $\mathcal{O}(N)$). In fact, the more obstacles in the environment, the less time each iteration takes. This is for two reasons: a. Obstacle cells do not require computation, and therefore are "skipped", and b. Obstacle cells have a "true" and constant value, therefore their neighboring cells converge to their own true values faster, as there are fewer variables in their surroundings. This effect propagates, making highly cluttered environments ideal for the method. This is why the method excels at "maze" type and cluttered environments.

The main drawback of the on-line navigation method is the computation complexity. The navigation method runs with a complexity of $\mathcal{O}(N^2)$, where N is the number of cells in the environment. If a robot needs to navigate in a very large space (say an office floor similar to Fig. 4, only much larger), the method may not be suitable in its current state. We also have not yet determined the upper bound on the number of iterations that guarantees the disappearance of local minima.

Comparison with other navigation methods: Naturally, we advocate the RRT method and its variants, that provide excellent results in many applications. Still, we offer an alternative method that shows promise for problems that are not easy to navigate using RRT, such as highly congested environments.

We emphasize that this paper deals with the question- *is this method fast enough for smooth navigation?* The results of the simulations and experiments show us that the answer is *yes*, and sometimes using faster robots than other methods can allow without pausing. By numerically solving the heat conduction equation with the obstacles as constantly hot points, and the target as a constantly cold one, the algorithm we have formulated has proven to be a robust variation of the potential function navigation method. Four inherent limitations of potential field navigation were presented in [8] that our algorithm seems to quell, namely trap situations due to local minima (as explained earlier), no passage between closely spaced obstacles (shown in Fig. 2), oscillations in the presence of obstacles and oscillations in narrow passages. The explanation for these improvements lies within the very different way the temperature gradient method works as opposed to classical potential fields. The target directly effects its environment, with that effect spreading towards all free space. If a point is reachable by connected space, it is guaranteed to be reached, eliminating the blockage of narrow passages. Oscillations do not occur as there are no "competing functions", as it is with traditional methods, thus the resulting gradient is generically smooth. This smoothness is

a physical attribute of a steady-state temperature gradient,and in practice we do not observe oscillations along the descending gradient.

Modern computers grant the luxury of using numeric solutions instead of analytic ones while remaining within reasonable time constraints for the robot's physical movement, thus allowing on-line navigation using sensors. However, as the method depends on grid size and has a complexity of $\mathcal{O}(N^2)$, large environments or higher dimensions could prove problematic.

We are currently working on faster methods for gradient convergence and dynamic obstacle handling capabilities. We are also considering the applicability of the method for multiple robot navigation. This seems a natural extension, as multiple robots can share the same temperature gradient, each one updating the gradient with new obstacles cells, and sharing the computational load to improve iteration time. Finally, we are also working on analytic solutions analogous to our numeric one. The superposition of point heat sources has an analytic solution, therefore the construction of point heat sources that can provide similar temperature gradients to those described in this paper may grant an analytic solution rather than a numerical one, thus allowing a much faster implementation.

REFERENCES

[1] G. Liu, W. Yu, and Y. Liu, "Resource management with RFID technology in automatic warehouse system," in *Proc. 2006 IEEE/RSJ Intl. Conf. Intell. Robots Syst*., 2006, pp. 3706–3711.

[2] L. P. Chuan, A. Johari, M. H. A. Wahab, D. M. Nor, N. S. A. M. Taujuddin, and M. E. Ayob, "An RFID warehouse robot," in *Proc. IEEE Intell. Adv. Syst.*, 2007, pp. 451–456.

[3] G. H. Lee, F. Faundorfer, and M. Pollefeys, "Motion estimation for self-driving cars with a generalized camera," in *Proc. IEEE Conf. Comput. Vis. Pattern Recogit.*, 2013, pp. 2746–2753.

[4] G. Silberg, R. Wallace, G. Matuszak, J. Plessers, C. Brower, and D. Subramanian, "Self-driving cars: The next revolution," *White paper, KPMG LLP Center Autom. Research*, 2012.

[5] H. R. Everett and D. W. Gage, "From laboratory to warehouse: Security robots meet the real world," *Int. J. Robot. Research*, vol. 18, no. 7, pp. 760–768, 1999.

[6] J. R. Andrews and N. Hogan, "Impedance control as a framework for implementing obstacle avoidance in a manipulator," M.S. thesis, Dept. Mech. Eng., MIT, Cambridge, MA, USA, 1983.

[7] E. Rimon and D. E. Koditschek, "Exact robot navigation using artificial potential functions," *IEEE Trans. Robot. Autom.*, vol. 8, no. 5, pp. 501–518, Oct. 1992.

[8] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Proc. IEEE Int. Conf. Robot. Autom.*, 1991, pp. 1398–1404.

[9] R. B. Tilove, "Local obstacle avoidance for mobile robots based on the method of artificial potentials," in *Proc. Robot. Autom.*, 1990, pp. 566–571.

[10] M. Wu, H. Ma, M. Fu, and C. Yang, "Particle filter based simultaneous localization and mapping using landmarks with RPLidar," in *Proc. IEEE Int. Conf. Intell. Robot. Appl*., Springer, 2015, pp. 592–603.

[11] M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Trans. Robot. Autom.*, vol. 17, no. 3, pp. 229–241, Jun. 2001.

[12] A. Sintov and A. Shapiro, "A stochastic dynamic motion planning algorithm for object-throwing," in *Proc. IEEE Int. Conf. Robot. Autom*., 2015, pp. 2475–2480.

[13] A. Sintov, O. Tslil, and A. Shapiro, "Robotic swing-up regrasping manipulation based on the impulse–momentum approach and CLQR control," *IEEE Trans. Robot.*, vol. 32, no. 5, pp. 1079–1090, Oct. 2016.

[14] C. I. Connolly, J. B. Burns, and R. Weiss, "Path planning using Laplace's equation," in *Proc. IEEE Int. Conf. Robot. Autom.*., 1990, pp. 2102–2106.

[15] S. Sasaki, "A practical computational technique for mobile robot navigation," in *Proc. IEEE Int. Conf. Control Appl.*, 1998, vol. 2, pp. 1323–1327.

[16] S. Garrido, L. Moreno, D. Blanco, and F. Martin, "FM2: A real-time fast marching sensor-based motion planner," in *Proc. IEEE/ASME Int. Conf. Adv. Intell. Mechatronics*., 2007, pp. 1–6.

[17] J. Holman, *Heat transfer, 8th SI Metric ed*. New York, NY, USA: McGraw-Hill, 2001.

[18] J. W. Thomas, *Numerical Partial Differential Equations: Finite Difference Methods* (Texts in Applied Mathematics 22). New York, NY, USA: Springer Science & Business Media, 2013, vol. 22.

[19] C. S. Peskin, "The immersed boundary method," *Acta Numerica*, vol. 11, pp. 479–517, 2002.

[20] A. Saudi and J. Sulaiman, "Path planning for indoor mobile robot using half-sweep SOR via nine-point Laplacian (HSSOR9l)," *IOSR J. Math.*, vol. 3, pp. 1–7, 2012.

[21] A. Saudi and J. Sulaiman, "Robot path planning using Laplacian behaviour-based control (LBBC) via half-sweep SOR," in *Proc. IEEE Int. Conf. Technological Adv. Electr., Electron. Comput. Eng.*, 2013, pp. 424–429.

[22] S. Garrido, L. Moreno, D. Blanco, and F. M. Monar, "Robotic motion using harmonic functions and finite elements," *J. Intell. Robot. Syst.*, vol. 59, no. 1, pp. 57–73, 2010.

[23] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," TR 98-11, Comput. Sci. Dept., Iowa State Univ., Oct. 1998.

[24] S. M. LaValle and J. J. Kuffner, *Algorithmic and Computational Robotics: New Directions*, B. R. Donald, K. M. Lynch, and D. Rus, Eds. Wellesley, MA, USA: A K Peters, 2001, pp. 293–308.

[25] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[26] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 2, 2000, pp. 995–1001.

[27] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with RRTs," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2006, pp. 1243–1248.

[28] S. R. Martin, S. E. Wright, and J. W. Sheppard, "Offline and online evolutionary bi-directional RRT algorithms for efficient re-planning in dynamic environments," in *Proc. IEEE Int. Conf. Autom. Sci. Eng*., 2007, pp. 1131–1136.

[29] O. Adiyatov and H. Varol, "Rapidly-exploring random tree based memory efficient motion planning," in *Proc. IEEE Int. Conf. Mechatronics Autom.*, 2013, pp. 354–359.

[30] L. Knispel and R. Matousek, RRT explorer, 2015. [Online]. Available: https://sites.google.com/site/rrtexplorer/

[31] SICK. Tim551-2050001 online datasheet, 2017. [Online]. Available: https://sick-virginia.data.continum.net/media/pdf/5/45/045/dataSheet_TIM551-2050001_1060445_en.pdf

[32] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, "Robust Monte Carlo localization for mobile robots," *Artif. Intell.*, vol. 128, no. 1, pp. 99–141, 2001.