A Light Formulation of the E* Interpolated Path Replanner

Roland Philippsen
Ecole Polytechnique Fédérale de Lausanne, Switzerland
roland.philippsen@gmx.net

June 21, 2006

Abstract

The E* algorithm is a path planning method capable of dynamic replanning and user-configurable path cost interpolation, it results in more appropriate paths during gradient descent. The underlying formulation is based on interpreting navigation functions as a sampled continuous crossing-time map that takes into account a risk measure. Replanning means that changes in the environment model can be repaired to avoid the expenses of complete planning. This helps compensating for the increased computational effort required for interpolation.

1 Introduction

Mobile robot path planning approaches can be divided into five classes [7]. Road map methods extract a network representation of the environment and then apply graph search to find a path. Exact cell decomposition methods construct non-overlapping regions that cover free space and encode cell connectivity in a graph. Approximate cell decomposition is similar, but cells are of predefined shape (e.g. rectangles) and do not exactly cover free space. Potential field methods [4] differ from the other four in that they treat the robot as a point evolving under the influence of forces that attract it to the goal while pushing it from obstacles. Navigation functions are free of local minima but otherwise similar to potential fields.

Grid or graph based methods such as NF1, A* [7], or Dijkstra's algorithm [1] can be considered to compute a navigation function constrained to movement choices along graph edges or discrete transitions between grid cells, they thus produce paths that are not optimal for execution. Frequently, a smoothing step is performed after planning to alleviate this problem, which yields unsatisfactory results as it only locally addresses a symptom rooted in the discrete nature of the path choices during planning.

In order to resolve this problem, we formulate navigation functions as a distance measure in the *continuous* domain, and devise an algorithm that computes samples of it. The latter are located at the nodes of a graph or grid which is embedded in the configuration space $\mathcal{C}$ of the robot [8]. The computation of the samples can be likened to graph search: It starts at the goal nodes and propagates through adjacent edges, assigning monotonically increasing values to nodes. The continuous formulation allows us to interpolate "between" edges, which resolves the issue with movement choices at a more fundamental level than methods that employ post-planning path smoothing.

2 Related Work

The weighted region path planning problem [9,11] is an approach to produce topologically correct paths that trade off movement cost against path length. However, the underlying environment representation is not expected to change, as needed e.g. for planning in dynamic environments such as mass exhibitions [3] or partially explored outdoor areas where regions are neither static nor known beforehand.

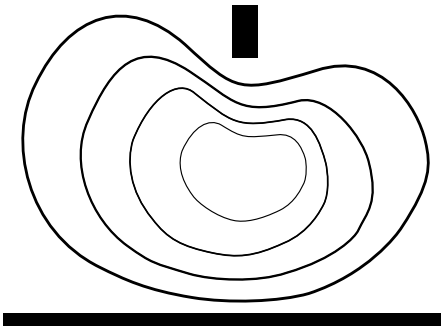


Figure 1: Continuous domain wavefront formulation. A contour sweeps outward from the goal throughout the environment, taking into account obstacle information. The crossing-time map can be interpreted as a navigation function: Following its steepest downward gradient will create an optimal path to the goal.

D* [6, 13] is graph based, suffering from the drawbacks mentioned in the introduction. Field-D*, a recent extension [2], uses linear interpolation to produce paths that are not constrained to edges. In many respects, it is similar[1] to E*, which was originally developed in [10] using an event-based formulation and has been replaced by a more "classical" procedure.

3 Overview of the E* Approach

The concept underlying the E* algorithm is formulated in the continuous domain independently of any specific $\mathcal{C}$-space representation. Consider a wavefront that propagates from the goal throughout the environment (see figure 1): If we record the front's evolution, we obtain a crossing-time map that is zero at the goal and monotonically increasing. This makes it suitable for gradient descent to find the goal from any point in the environment. By modulating the propagation speed in function of environment characteristics, i.e. the effort of traversing certain regions, we can create a crossing time map that has a steeper gradient when close to obstacles.

The monotonicity of the crossing-time map stems from the upwind property: The wavefront can be interpreted as the envelope of all possible locations the robot can reach if it starts on the goal and always moves with the maximum allowed speed. Like grass fire, this envelope is ever expanding, which allows us to uniquely determine the region of influence of each location by tracing the propagation direction. If a location is subsequently modified, it is thus possible to determine which portions of an existing crossing-time map can be kept intact, and which portions need re-computation. This is the key to efficient replanning by reducing the effort of incorporating new environmental information.

4 The E* Algorithm

As a method for calculating interpolated navigation functions on graphs (or grids) embedded in a continuous $\mathcal{C}$-space, E* can locally repair an existing navigation function. The elements that are used to achieve this are presented in the following: A common definition of graph-based environment models, an ordered queue to ensure appropriate propagation, tracing upwind information for correctly initializing path cost changes, conditions for interpolation kernels, and a mechanism for determining the set of nodes that can influence the interpolation.

[1] The differences and similarities between Field-D* and E* will be discussed in an upcoming collaborative paper.

4.1 Environment Representation

The environment in which the robot evolves is represented as an undirected graph G embedded in configuration space $\mathcal{C}$. Note that grids fall into this category. Several properties are attached to the nodes $c \in G$, two of which are relevant for a high-level understanding:

- The *value* $v(c) \geq 0$ represents the sample of the continuous crossing-time map at the node c, or the "height" of the navigation function. This is similar to the optimal path costs of A* or the cell labels of NF1.
- The difficulty or cost of traversing a given configuration is encoded in the traversal *effort* or risk $r(c) \in [0, 1]$. A lower r implies a higher speed. Effort is converted to *meta* information $m(c)$ to allow for different interpolation kernels, as will be explained below.

Many graph-based planners attach effort (path cost) to the *edges* of a graph, contrary to the approach taken here. Our choice is based on the rationale that nodes represent regions of the environment, and edges encode topological information by linking regions that are adjacent to each other. The robot spends energy for traversing the regions (i.e. nodes), whereas edges are transitions in the robot's representation of localization knowledge, their "traversal" does not incur movement costs.

4.2 Wavefront Propagation

As in A*, where it is called OPEN list, the wavefront is a queue of nodes that await *expansion*, which is the elementary propagation step from a given node to its neighbors. Planning proceeds until the wavefront is empty or the node "containing" the robot has been expanded. The wavefront is ordered by ascending *key*, which is designed to result in a strictly upwind propagation order. Similarly to D*-Lite [6], a one-step lookahead of the crossing time called rhs-value is used in conjunction with the estimated $v(c)$ to calculate the queue key as $\min(v(c), \mathrm{rhs}(c))$. When the rhs and value of a node equal each other, the node is called (locally) consistent. Wavefront propagation drives the algorithm towards a state where all nodes are locally consistent.

4.3 Upwind Graph

A* uses a spanning tree to trace the path from robot position to goal. E* extends this to a directed upwind graph U with unique edges $(c_1, c_2) \in U \Rightarrow (c_2, c_1) \notin U)$. It allows to retrieve the upwind set $U(c)$ of nodes that were involved in computing $v(c)$ as well as the downwind set $D(c)$ of nodes that were influenced by $v(c)$ during interpolation of their value. Thus, all descendants of a node can be determined if its environmental information subsequently changes, similarly to the backpointers in D*.

4.4 Interpolation Kernel

An interpolation kernel k, formally defined in (1), is a function that estimates the crossing-time value of a node, based on the risk (or effort) of traversing it, in conjunction with the values of its neighbors.

$$(u, B) = k(c, Q) \tag{1}$$

where u is the new value for node c, $B \subseteq Q(c)$ is the set of neighbors used in the computation of u, and $Q(c)$ is the propagator of c at the time of expansion (2), which ensures that only valid candidate neighbors are provided to the kernel.

$$Q(c) = \{n \in N(c) \mid \ v(n) < \infty\} \tag{2}$$

where $N(c)$ is the set of neighbors (adjacent nodes) of c. Nodes with infinite value are either obstacles or have not been expanded yet, such as after initialization or following an increase in

$r(c)$. Q is (partially) ordered by ascending v of the contained nodes: $v(Q_i) \leq v(Q_j)\ \forall\ i < j$ where Q_i denotes its i^{th} entry.

It is possible for Q to be empty ($Q = \{\}$) or contain a single node ($Q = \{Q_1\}$). Handling these cases is part of the required properties for a kernel (3).

$$\begin{cases} \text{(a)} & u(c, Q) > v(Q_1) \\ \text{(b)} & u(c, Q) \leq u(c, \{Q_1\}) \\ \text{(c)} & u(c, Q = \{\}) = \infty \end{cases} \tag{3}$$

where (a) is the upwind condition that ensures monotonic propagation, (b) signifies that using interpolation yields a lower value than taking a discrete movement choice (which might be necessary as fallback solution even in the presence of more than one propagator), and (c) formalizes that it is not valid to propagate unless at least one neighbor is a non-obstacle node which is known to be reachable.

Another important condition, which is not formalized in (3), is that B must accurately reflect the information used for the computation of u. This is required such that modifications to the environment model can be consistently propagated to all concerned nodes.

4.5 The Algorithm

Listing 1 shows pseudo-code for E*. $\{g_i\}$ is the set of goal nodes, W denotes the wavefront queue, and key(c) is the key with which c has been inserted into W. Note that $D(c)$ in line 23 has to be copied, because the call to UpdateVertex(d) changes the upwind graph U. Procedure ComputePropagator(c) applies equation (2), Pop(W) removes and returns the top node from W, and TopKey(W) returns the key of the top node or ∞.

5 The LSM Kernel

Early development of E* was closely linked with an adaptation of the *Level Set Method* (LSM) [12] to mobile robot path planning. The LSM provides a robust grid-based algorithm for calculating the time-dependent position of an evolving curve. The results of this adaptation are available as the LSM kernel k_{LSM} in E*. It is applicable when the $\mathcal{C}$ graph G is a four-connected regular grid. This section starts with an introductory summary of the LSM and the *Fast Marching Method*, a special case formulation applicable to path planning [5]. These summaries are necessary for understanding the expressions for k_{LSM} presented afterward.

5.1 The Fast Marching Level Set Method

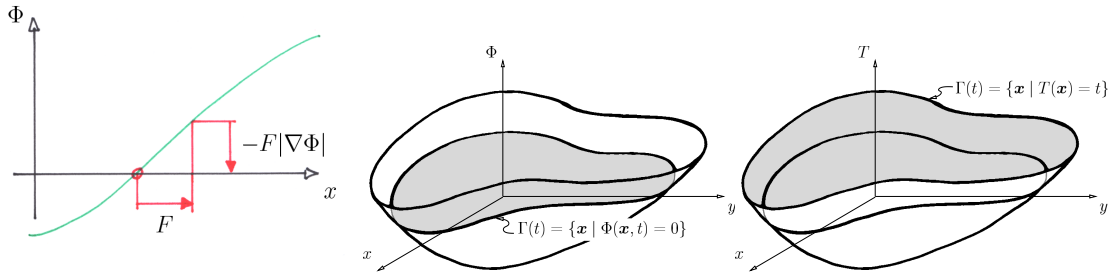
The attempt to determine the evolution of a front such as the one in figure 1 by using a parameterized curve, deriving its normal vector, and moving discrete points of the curve along this normal, is called the *Lagrangian* formulation. It presents several drawbacks [12]. A solution lies in using a *Eulerian* formulation, which adds a dimension to the problem and then defines the wavefront as the intersection between a surface and the zero-level of the additional dimension. This is illustrated in figure 2(b) and equation (4).

$$\begin{cases} \Gamma(t): & \text{closed } (N-1)\text{D surface} \\ \Phi(\vec{x}, t): & \mathbb{R}^N \rightarrow \mathbb{R} \\ t_0: & \Phi(\vec{x}, t = 0) = \pm d(\vec{x}, \Gamma(t = 0)) \\ \Rightarrow & \Gamma(t) = \{\vec{x} \mid \Phi(\vec{x}, t) = 0\} \end{cases} \tag{4}$$

where Γ denotes the wavefront at instant t, N is the supporting dimension, and Φ is the surface that is intersected with the zero level to yield Γ. The line labeled t_0 indicates that Φ is initialized to the signed distance from the initial wavefront.

Listing 1 Pseudo code of the core procedures in E*.

```
procedure Requeue(c)
01  if v(c) = rhs(c)
02    if c ∈ W then remove c from W
03  else
04    if c ∉ W
05      insert c with key = min(v(c), rhs(c)) into W
06    else if key(c) ≠ min(v(c), rhs(c))
07      remove c from W
08      insert c with key = min(v(c), rhs(c)) into W
procedure UpdateVertex(c)
09  if c ∉ {g_i}
10    Q ← ComputePropagator(c)
11    (rhs(c), B) ← k(c, Q)
12    for all u ∈ U(c) remove (u, c) from U
13    for all b ∈ B
14      if (c, b) ∈ U then remove (c, b) from U
15      add (b, c) to U
16    Requeue(c)
procedure Propagate()
17  c ← Pop(W)
18  if v(c) > rhs(c)
19    v(c) ← rhs(c)
20    for all n ∈ N(c) UpdateVertex(n)
21  else
22    v(c) ← ∞
23    for all d ∈ D(c) UpdateVertex(d)
24    UpdateVertex(c)
procedure main()
25  initialize rhs(c) = v(c) = ∞ ∀c ∈ G
26  initialize goal rhs(g) =true distance ∀g ∈ {g_i}
27  initialize W with {g_i}
28  while (rhs(c_robot) ≠ v(c_robot)) or TopKey(W) < v(c_robot)
29    if W = {} then the goal is unreachable
30    Propagate()
```



(a) 1D case (b) Eulerian (c) Eikonal

Figure 2: In the Eulerian perspective, the wavefront is interpreted as the intersection between a graph (b) and the zero-level of an additional dimension Φ. (a) shows the one-dimensional case of the Eulerian formulation to clarify how equation (5) describes the wavefront's evolution: In order to make the intersection move towards the right with speed F, the whole curve Φ has to move downwards with speed $F|\nabla\Phi|$. The Eikonal case (c) leads to a simplified formulation of the Level Set Method that can be solved very efficiently. Notation has changed from Φ to T.

The advantage of adding an extra dimension is that topology changes (e.g. merging of a front after propagation around an obstacle) do not require special treatment, and that numerically stable methods are available for solving the differential equation that describes the front's evolution. This is illustrated in figure 2(a) and equation (5).

$$\frac{\partial \Phi}{\partial t} + F|\nabla \Phi| = 0 \tag{5}$$

where F is the propagation speed. The LSM can be summarized as follows: Convert the initial wavefront $\Gamma(t=0)$ into a surface $\Phi(\vec{x}, t=0)$ by taking the signed distance from $\vec{x}$ to the initial front; repeatedly solve equation (5); determine the front's evolution $\Gamma(t)$ by intersecting $\Phi(t)$ with the zero level. This requires a discrete approximation of the gradient operator ∇, a finite timestep Δt, as well as some numerical adaptations to make it robust [12].

The Fast Marching Method can be used when the propagation speed is always positive (or negative) and depends on position only, i.e. the path planning problem we want to solve. This is called the *Eikonal* case, which allows efficient computations: It treats Φ as a crossing-time map and $\Gamma(t)$ is obtained by intersecting Φ with the level of height t. In order to stress this difference, a notational change is introduced: Φ becomes T. Figure 2(c) and equation (6) illustrate the Eikonal case.

$$\begin{cases} F = F(\vec{x}) > 0 \\ \Gamma(t) = \{\vec{x} \mid T(\vec{x}) = t\} \\ |\nabla T| F = 1 \end{cases} \tag{6}$$

where F denotes the propagation speed and the wavefront Γ is the intersection between the crossing-time map T and a given instant t. Now, T can be built outward starting at $T = 0$. This is due to the upwind property, a location is traversed only once by the wavefront, because $F \geq 0$.

Given that the LSM is calculated on a grid, a discrete notation is introduced: $T(\vec{x})$ becomes T_ι with ι an $\mathcal{N}$-dimensional index, $F(\vec{x})$ becomes F_ι, and ∇ is replaced by the finite difference $D^{\pm x_i}$ with x_i an axis of the grid, resulting in (7) for numerically solving $|\nabla T| F = 1$.

$$\sum_{i=1}^{\mathcal{N}} \left(\max(D^{-x_i} T_\iota, 0)^2 + \min(D^{+x_i} T_\iota, 0)^2\right) = \frac{1}{F_\iota^2} \tag{7}$$

The Fast Marching Method incrementally solves this equation throughout the grid. This resembles E* but does not provide replanning. By translating (7) into k_{LSM}, the advantages of the LSM approach can be combined with the replanning capabilities of E*.

5.2 LSM Kernel Based on Fast Marching

Here we present a two-dimensional implementation of k_{LSM} using a first-order upwind interpolation scheme for (7). It is trivial to add more dimensions. Equation (8) gives the expressions for $D^{\pm x_i}$.

$$\begin{cases} D^{-x} T_{ij} &= (T_{ij} - T_{i-1,j})/h \\ D^{+x} T_{ij} &= (T_{i+1,j} - T_{i,j})/h \\ D^{-y} T_{ij} &= (T_{ij} - T_{i,j-1})/h \\ D^{+y} T_{ij} &= (T_{i,j+1} - T_{i,j})/h \end{cases} \tag{8}$$

where (i, j) is the grid index, $x_1 = x$, $x_2 = y$, and h is the grid resolution (distance between two nodes along).

Developing $D^{\pm\{x,y\}} T_{ij}$ leads to a quadratic equation with coefficients that take values based on a switch on the sign and magnitude of the finite difference operators. A brute-force computation would enumerate all permutations, calculate their resulting T, and then choose the smallest valid one. However, it is possible to proceed in a more informed manner. A geometric interpretation helps to determine the terms that will yield the optimal solution *prior* to interpolating. Figure 3(a)

shows the situation, following the development in [5]. The cell in the center is being updated. Interpolation implies using up to two neighbors, which need to lie on different axes. Without loss of generality, it can be assumed that the two neighbors leading to the best interpolation are **A** and **C**, and that $T_\mathrm{A} \leq T_\mathrm{C}$. The update equation becomes (9).

$$(T - T_\mathrm{A})^2 + (T - T_\mathrm{C})^2 = h^2/F^2$$
$$\Leftrightarrow \begin{cases} T = t_\mathrm{A} = t_\mathrm{C} \\ (t_\mathrm{A} - T_\mathrm{A})^2 + (t_\mathrm{C} - T_\mathrm{C})^2 = h^2/F^2 \end{cases} \tag{9}$$

where T is to be determined, T_A and T_C are the values of the best neighbors, h is the distance between two neighbors, and F is the propagation speed at (i, j).

Figure 3(b) shows the overall geometrical interpretation for a given $(T_\mathrm{C}, T_\mathrm{A})$. The two parameters t_C and t_A are interpreted as the axes of a Cartesian coordinate frame. The solutions for (9) are found at the intersections between the diagonal $t_\mathrm{A} = t_\mathrm{C}$ and a circle of radius h/F centered at $(T_\mathrm{C}, T_\mathrm{A})$.

The max and min operators surrounding $D^{\pm x_i} T_\iota$ in (7) lead to constraints that need to be added to (9): Either it has a real solution T with $T > T_\mathrm{C}$, or a real solution to the degenerate form (10) with $T_\mathrm{A} < T \leq T_\mathrm{C}$. In figure 3(b), the degenerate (fallback) solution is equivalent to the intersection between a horizontal line $t_\mathrm{A} = T_\mathrm{A} + h/F$ and the diagonal $t_\mathrm{A} = t_\mathrm{C}$.

$$(T - T_\mathrm{A})^2 = \frac{h^2}{F^2} \;\Leftrightarrow\; \begin{cases} T = t_\mathrm{A} = t_\mathrm{C} \\ t_\mathrm{A} = T_\mathrm{A} + h/F \end{cases} \tag{10}$$

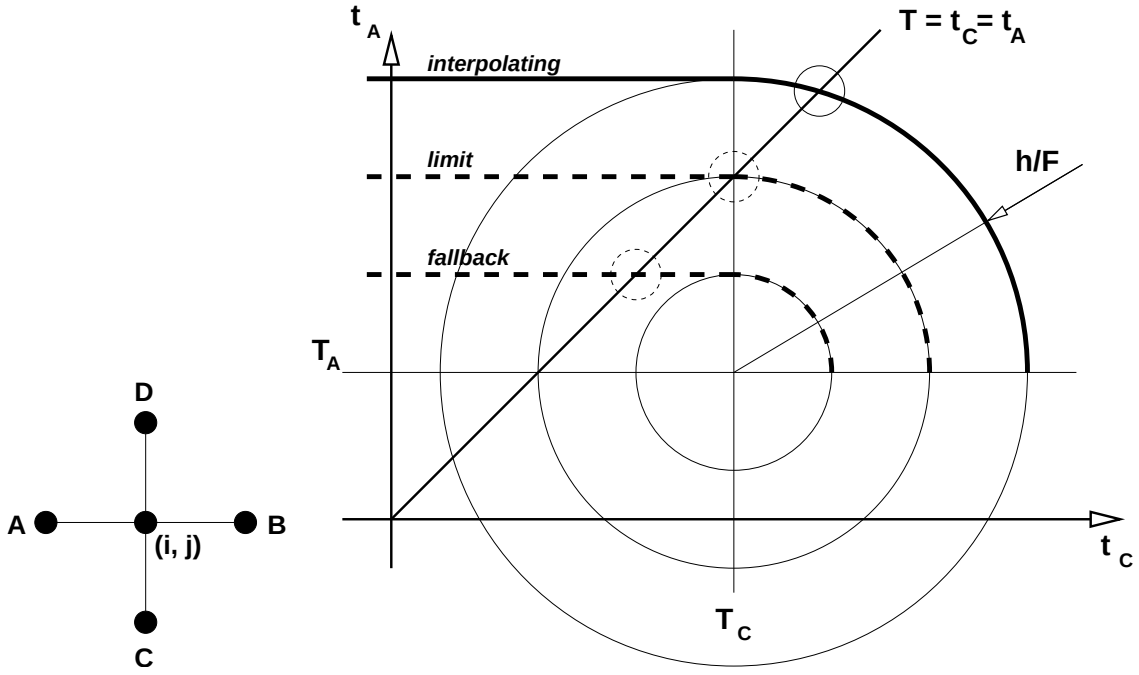
Equation (9) has to be solved only if the point $(T_\mathrm{C}, T_\mathrm{A} + h/F)$ lies above $t_\mathrm{C} = t_\mathrm{A}$ (i.e. the *interpolating* curve in figure 3(b)), and only the higher of the two intersections has to be computed (11). The final equations for k_LSM are given in (12).

$$\begin{aligned} T &= \begin{cases} T_\mathrm{A} + h/F & \Leftarrow\; T_\mathrm{C} - T_\mathrm{A} \geq h/F \\ \frac{1}{2}\left(-\beta + \sqrt{\beta^2 - 4\gamma}\right) & \text{otherwise} \end{cases} \\ \beta &= -\left(T_\mathrm{A} + T_\mathrm{C}\right) \\ \gamma &= \tfrac{1}{2}\left(T_\mathrm{A}^2 + T_\mathrm{C}^2 - h^2/F^2\right) \end{aligned} \tag{11}$$

$$(u, B) = k_\mathrm{LSM}(c, Q)$$
$$\begin{aligned} u &= T \\ B &= \begin{cases} \{Q_1\} & \Leftarrow\; T_\mathrm{C} - T_\mathrm{A} \geq h/F \\ \{Q_1, Q_2\} & \text{otherwise} \end{cases} \\ T_\mathrm{A} &= v(Q_1) \\ T_\mathrm{C} &= \begin{cases} \infty & \Leftarrow\; Q = \{Q_1\} \\ v(Q_2) & \text{otherwise} \end{cases} \\ F &= m(c) \end{aligned} \tag{12}$$

recall that $T_\mathrm{A} \leq T_\mathrm{C}$ and note that $F \to 0 \;\Rightarrow\; T \to \infty$. Also note that nodes which do not have neighbors of type **A** and **C**, such as neighbors of obstacles or cells on the grid border, use the fallback solution $T_\mathrm{A} + h/F$.

The LSM kernel maps zero effort $r = 0$ to the maximum propagation speed $F_{ij,\max} = 1$, and obstacles $r = 1$ to $F_{ij} = 0$. Thus, the meta information for k_LSM is the propagation speed: $m(c) = F(c)$. Note that choosing a unit $F_{ij,\max}$ conveniently yields the distance to the goal in environments that can be modeled as either pure free-space or obstacle.



(a) situation

(b) geometrical interpretation

Figure 3: Geometric interpretation of LSM interpolation, (a) shows the node neighborhood. Equations (9) and (10) can be read as finding the intersection between the line $t_C = t_A$ and the thick solid curve labeled *interpolating* in (b). Two dashed curves illustrate how the interpolation behaves when h/F becomes smaller (*limit* and *fallback* curves). The small solid circle indicates the intersection that serves as solution for the interpolating case, and the small dashed circles show the same for the limit and fallback cases.

Zig-zag map			
cell size	0.67	0.37	0.20
k_{NF1}			
N° dyn. propagations	1'242	3'926	13'627
N° replan prop.	2'209	7'279	26'226
gain	**43.8** %	**46.1** %	**48.9** %
k_{LSM}			
N° dyn. propagations	1'759	6'417	20'260
N° replan prop.	2'243	7'337	26'192
gain	**21.6** %	**12.5** %	**22.6** %
Maze map			
cell size	0.71	0.38	0.20
k_{NF1}			
N° dyn. propagations	3'521	11'890	43'836
N° replan prop.	6'881	25'463	95'075
gain	**48.8** %	**53.3** %	**53.9** %
k_{LSM}			
N° dyn. propagations	5'464	18'887	74'880
N° replan prop.	8'565	26'880	116'271
gain	**36.2** %	**29.7** %	**35.6** %

Table 1: E* replanning performance

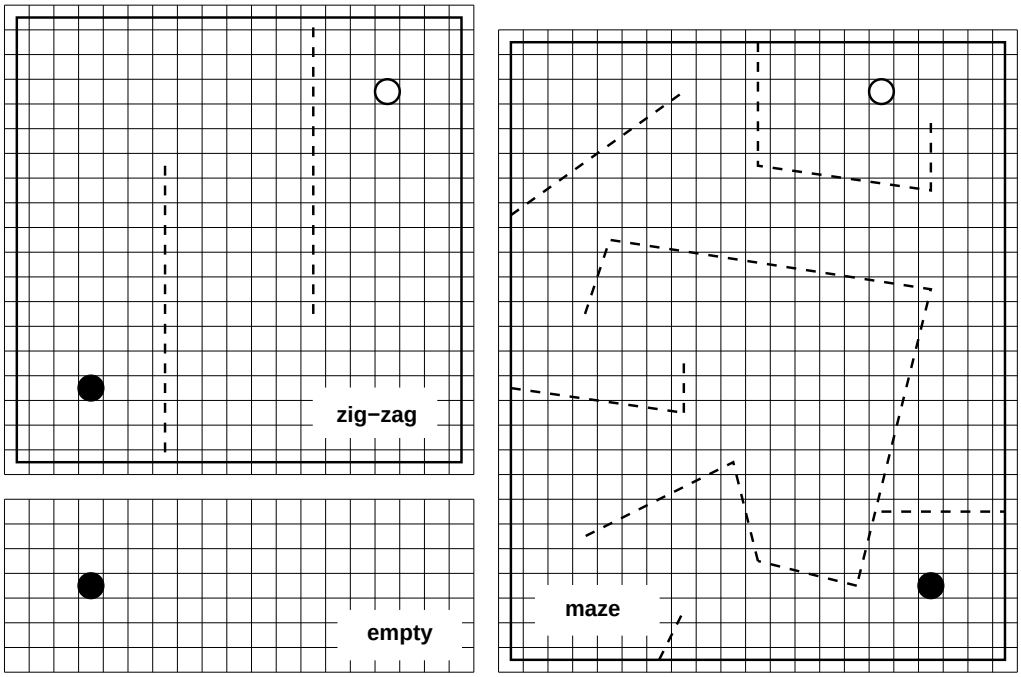
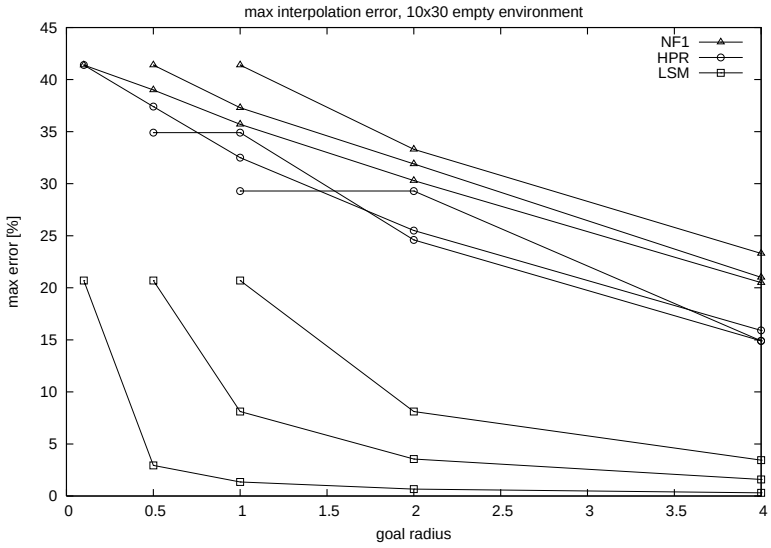


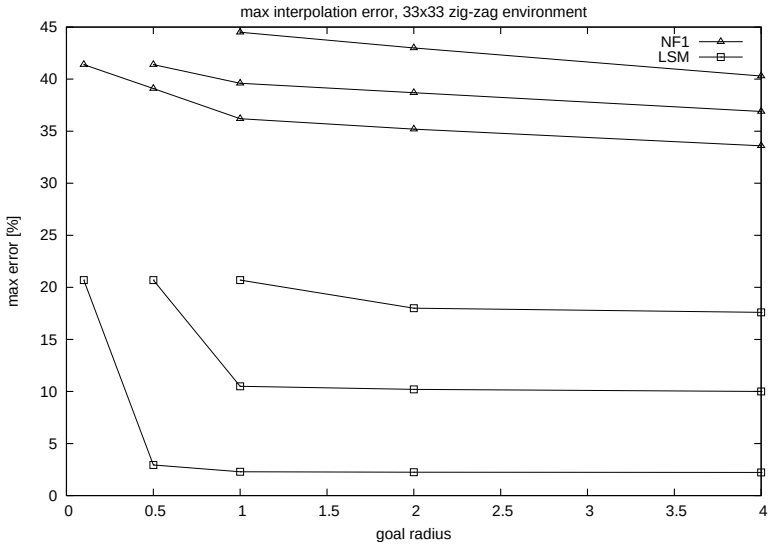
Figure 4: Evaluation environments: Solid lines denote a-priori known obstacles, dotted lines indicate obstacles that need to be discovered by the robot during its movement. The empty circle is the start, and the solid circle the goal. The empty environment is used for determining in which way the initial goal radius influences the convergence towards the true Euclidean distance. The zig-zag and maze environments are used to determine the gain from dynamic replanning and to compare the path smoothness produced by different interpolation methods. The grid resolution was higher than shown during simulations.

cell size	0.71	0.38	0.20
Zig-zag map			
relative burden ρ(NF1)	0.562	0.539	0.520
relative burden ρ(LSM)	0.784	0.875	0.774
ρ(LSM)/ρ(NF1)	1.39	1.62	1.49
Maze map			
relative burden ρ(NF1)	0.512	0.467	0.461
relative burden ρ(LSM)	0.638	0.703	0.644
ρ(LSM)/ρ(NF1)	1.25	1.50	1.40

Table 2: E* relative computational complexity.



(a) $\max e$ in 10×30 empty environment



(b) $\max e$ in 33×33 zig-zag environment

Figure 5: Relative error between E* and true distance. The maximum error e is plotted in function of the kernel and the goal radius $\{0.1, 0.5, 1, 2, 4\}$. k_{LSM} with $e \in [0.302\%, 20.7\%]$ performs better than k_{HPR} and k_{NF1} with $e \in [20.5\%, 41.1\%]$. The three lines per interpolation method correspond to the three cell sizes: $h = 1$ at the top, $h = 0.5$ in the middle, and $h = 0.1$ at the bottom. The overall heightened errors of (b) with respect to (a) are due to the difference between ground truth calculation using the exact endings of the walls, and the propagation, which goes through the first non-occupied cell near the end of each wall.

6 Results

The precision of E* has been tested in comparison to ground truth distances in completely known environments, and replanning consistency and efficiency gains have been measured in simulated exploration of an initially unknown environment. To quantify the effects of interpolation, the NF1 kernel k_{NF1} is introduced. It mimics the NF1 algorithm. The update equation (13) is straightforward, and the meta information is $m(c) = 0$ for free space and $m(c) = \infty$ for obstacles.

$(u, B) = k_{\mathrm{NF1}}(c, Q)$ with

$$u = \begin{cases} \infty & \Leftarrow Q = \{\} \\ \min_{q \in Q} (v(q) + h + m(c)) & \text{otherwise} \end{cases}$$
$$B = \begin{cases} \{\} & \Leftarrow u = \infty \\ \{\arg\min_{q \in Q} (v(q) + h + m(c))\} & \text{otherwise} \end{cases} \tag{13}$$

Figure 4 shows the experimental set-ups. Cells within circular goal region are initialized to the Euclidean distance to the goal point. First we measured how the kernel, the resolution, and the goal radius influence the relative error $e = (v(c) - d(c))/d(c)$, where $d(c)$ is the true distance from c to the goal. Figure 5 shows the maximum values of e for the empty and zig-zag environments. None of the kernels under-estimates the distance to the goal, all improve e when increasing the ratio of goal radius over cell size. Note that the first run in each series initialized using a single goal cell and thus indicates the effects of fallback solutions.

Second, we determined the amount of work saved by replanning, which depends on the environment and the update frequency. We simulated a point robot descending the gradient (see figure 6) with bounds on acceleration and speed, equipped with a sensor of limited range. When a new obstacle is detected, propagation is performed until the robot node has a value lower than TopKey(W), counting the number of propagations.

Table 1 lists replanning performance: The gain from dynamic over complete replanning is lessened for k_{LSM}, cell size is not very significant. **N° dyn. propagations** is the expansion count over the whole run when using replanning; **N° replan prop.** is the event count over the whole run when using re-initialization and planning from scratch; **gain** is the relative reduction in the number of events, calculated as $(n_{\mathrm{complete}} - n_{\mathrm{dynamic}})/n_{\mathrm{complete}}$.

Table 2 compares the propagation counts. $\rho = n_{\mathrm{dynamic}}/n_{\mathrm{complete}}$ normalizes the number of dynamic replanning events. $\rho(\mathrm{LSM})/\rho(\mathrm{NF1})$ indicates how much wider raise events spread when using interpolation. The theoretical increase is twofold (twice as many backpointers), however the observed values lie between a 25% and 62% increase. However, that the operation cost for one step in k_{LSM} is up to 37% higher than for k_{NF1} (details not shown here).

Figure 6 shows the smoothness of the paths produced with k_{LSM} during the runs that served to collect the above data. They are close to the line-of-sight towards the edge of an obstacle (if discovered) or the goal.

7 Conclusion and Outlook

The E* algorithm allows to calculate and update smooth navigation functions that approximate true distance much better than other grid or graph based methods – by an order of magnitude or more given the right conditions. The additional computational complexity required for this achievement is a factor of two in the theoretical worst case, but experiments suggest a factor of approximately 1.2 to 1.6 in practice. Each of the propagation steps becomes more elaborate as well, this amounts to a factor below 1.4 in the case of the robust LSM interpolation when compared with NF1.

The core E* algorithm is similar to D*-Lite, which is capable of focused heuristic search to speed up planning. This is feasible for E* as well. However, interpolation introduces a complication

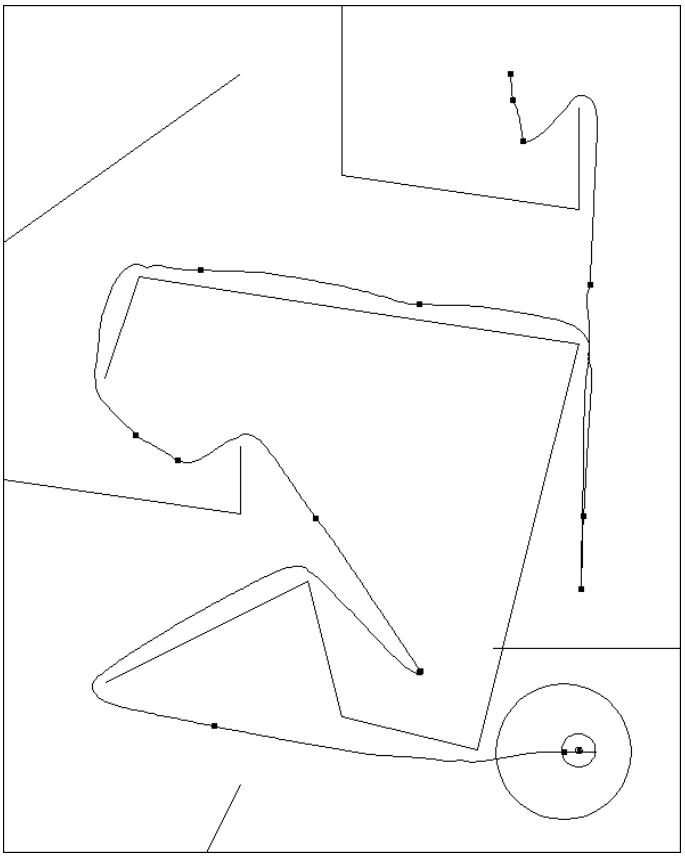


Figure 6: Paths from exploring the maze map using k_{LSM}, the robot starts in the upper right without any knowledge of the environment and proceeds to reach the goal in the lower right, discovering obstacles that come into its sensor range. Thick dots are places where new obstacle information was incorporated.

that has not been resolved yet: When the node containing the robot is made consistent with a value below the smallest queue key, this does not mean that the best interpolated route has been found. In fact, interpolation will arrive "a bit later", and we are working on a robust method of figuring out when this has occurred.

Implementing k_{LSM} in higher dimensions or using higher-order interpolation is possible. Extending it to non-grid representation is also feasible, the LSM has been applied to triangulated domains in [5].

In addition to providing an interpolated navigation functions with dynamic replanning, E* is independent of interpolation details and can thus be used to evaluate different kernels in terms of their quality and computational costs.

Acknowledgment

The authors would like to thank Kurt Konolige for pointing out the Level Set Method, and Dave Ferguson for pointing out D*-Lite as well as his support and many worthwhile discussions.

References

[1] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[2] Dave Ferguson and Anthony Stentz. Field D*: An interpolation-based path planner and replanner. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, 2005.

[3] B. Jensen, R. Philippsen, and R. Siegwart. Motion detection and path planning in dynamic environments. In *Proceedings of the Workshop on Reasoning with Uncertainty in Robotics, International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

[4] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5(1), 1986.

[5] R. Kimmel and J.A. Sethian. Computing geodesic paths on manifolds. *Proc. Natl. Acad. Sci. USA*, 95(15):8431–8435, July 1998.

[6] S. Koenig and M. Likhachev. D* lite. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2002.

[7] J.-C. Latombe. *Robot motion planning*. Kluwer Academic Publishers, Dordrecht, Netherlands, 1991.

[8] Tomás Lozano-Perez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, 32(2):108–120, February 1983.

[9] J. S. B. Mitchell and C. H. Papadimitriou. The weighted region problem: Finding shortest paths through a weighted planar subdivision. *Journal of the ACM*, 38(1):18–73, 1991.

[10] Roland Philippsen. *Motion Planning and Obstacle Avoidance for Mobile Robots in Highly Cluttered Dynamic Environments*. PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 2004.

[11] N. C. Rowe and R. S. Alexander. Finding optimal-path maps for path planning across weighted regions. *International Journal of Robotics Research*, 19(2):83–95, 2000.

[12] J.A. Sethian. *Level Set Methods – Evolving interfaces in geometry, fluid mechanics, computer vision, and materials science*. Cambridge University Press, 1996.

[13] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 1994.