E* Interpolated Graph Replanner

Roland Philippsen
Stanford University Robotics Laboratory
roland.philippsen@gmx.net
http://estar.sourceforge.net/

October 29, 2007

Abstract

The E* algorithm is a path planner which supports dynamic replanning and path cost interpolation, resulting in lightweight repairing of plans and smooth paths during execution. Underlying E* is the interpretation of navigation functions as the crossing-time map of an expanding continuous surface whose propagation takes into account traversability.

Unlike A* which constrains movements to graph edges, E* produces smooth trajectories by interpolating between edges. Like D* it supports dynamic replanning after local path cost changes. Similarly to Field-D* it uses interpolation to obtain smoothly varying values. However, E* interpolation is user-configurable, and it performs full tracking of the upwind dependency structure.

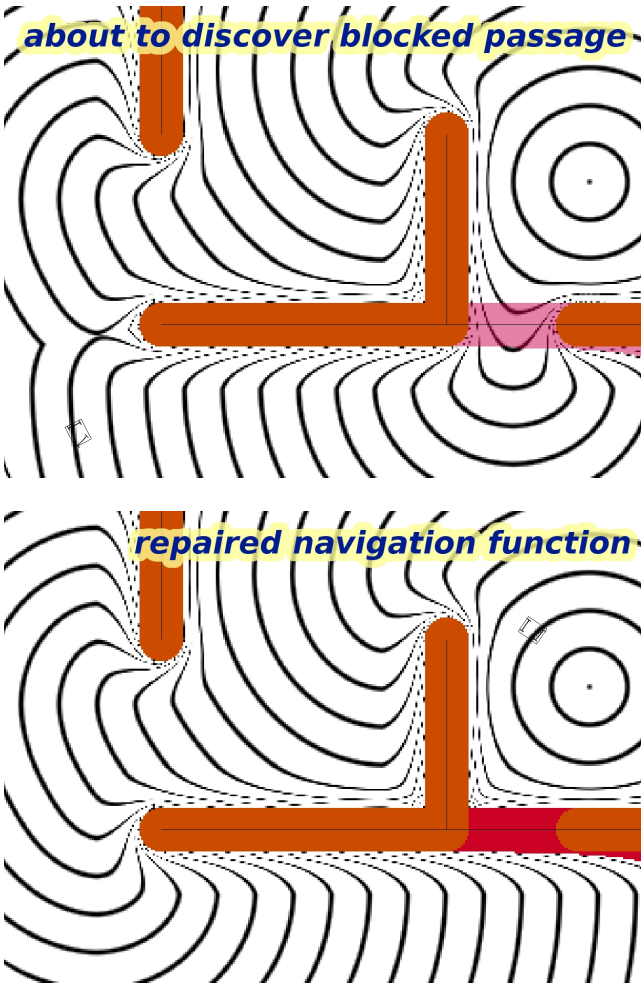
An open-source implementation of the algorithm is available on *Sourceforge* along with several test programs and examples of integration into robotic systems.

Introduction

- Classical computations of navigation functions [1, 7] constrain movement to graph edges ⇒ impractical for path execution. Potential fields [3] are smooth and continuous, but can have local minima. Address root of problem [10, 11] by considering navigation functions as a distance measure in the *continuous* domain: E* computes samples of this distance by propagating through a graph embedded in $\mathcal{C}$-space [8], interpolating between edges to assign monotonically increasing values to nodes.

- Environment information evolves ⇒ a planner must efficiently handle traversability changes. Purely graph-based D* [5, 14] performs well, Field-D* [2] extends it using somewhat ad-hoc hard-coded linear interpolation. E* incorporates a similar capability of path repairs by propagating changes out from changed locations, and additionally supports a generic interpolation formulation that allows user-defined kernels.

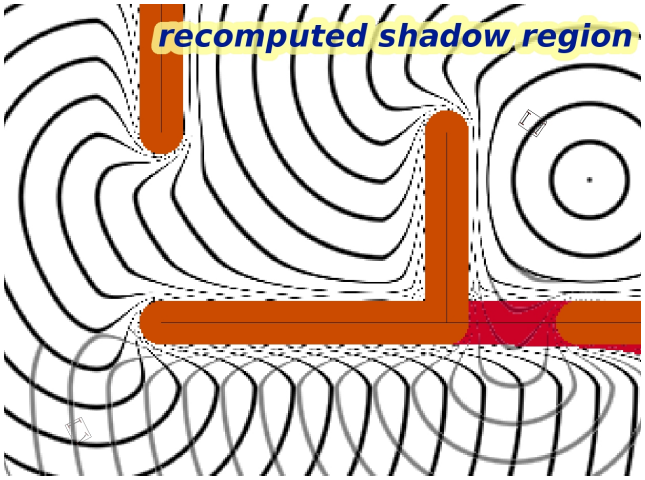
- Further reading: weighted region path planning [9, 12] (trade-off movement cost against path length given traversability that does not change), gradient method [6] (alternative interpolation approach without path repairs but fast enough to replan from scratch).



Features

Smoothness: A continuous-domain wavefront (closed surface) propagates from the goal through the environment. Modulate propagation speed in function of environment characteristics. Record the front's evolution ⇒ monotonically increasing crossing-time map suitable for gradient descent from any point to the goal.

Local repairs: the ever-expanding wavefront determines the region of influence of each location ⇒ when subsequently modified, we can skip recomputing those regions that were never influenced by the modified location.



Components

Representation: undirected graph G embedded in $\mathcal{C}$-space. Properties are attached to the nodes $c \in G$: *(i)* the *value* $v(c) \geq 0$ of the crossing-time map; *(ii)* the effort of traversing a configuration (encoded in the *risk* $r(c) \in [0, 1]$); converted to *(iii) meta* information $m(c)$ to allow for different interpolation kernels; and *(iv)* the rhs(c) is a one-step lookahead of the crossing time (see D*-Lite [5]).

Wavefront: queue of nodes that await *expansion* (value propagation from a given node to its neighbors), ordered by a *key* ensuring strictly upwind propagation order: *key* $= \min(v(c), \text{rhs}(c))$.

Upwind Graph: extension of the A* spanning tree to a directed upwind graph U with unique edges $(c_1, c_2) \in U \Rightarrow (c_2, c_1) \notin U$, encodes the nodes that were involved in computing $v(c)$ as well as the ones that were influenced by $v(c)$ similarly to the backpointers in D*.

Generic Interpolation: a kernel $(u, B) = k(c, Q)$ estimates the crossing-time value of a node, based on the risk of traversing it and the values of its neighbors: u is the new value for node c, $B \subseteq Q(c)$ is the set of neighbors used in the computation of u, and $Q(c) = \{n \in N(c) \mid v(n) < \infty\}$ is the propagator which ensures that only valid candidate neighbors $N(c)$ are provided to the kernel.

LSM Interpolation Kernel

The LSM kernel k_{LSM} is an adaptation of the *Fast Marching Level Set Method* [4, 13] applicable when the $\mathcal{C}$ graph G is a four-connected regular grid. In the 2D case this leads to a quadratic expression that provides smooth, robust and lightweight interpolation. The LSM kernel maps zero effort $r(c) = 0$ to the maximum propagation speed $m(c) = F_{\max} = 1$, and obstacles $r(c) = 1$ to zero speed $m(c) = 0$.

Open Source Implementation

E* is implemented in C++ and released under the GNU General Public License on `http://estar.sourceforge.net/` with access to the source code repository, mailing lists, documentation, and a Wiki.

Supported Operating Systems: all POSIX (Linux, BSD, and other UNIX-like OSes). To a large extent, the code is OS-independent and can easily be ported to other systems provided that the dependencies are available there as well.

Dependencies: C++ Standard Library and Standard Template Library, Boost Graph Library, Boost Smart Pointers, GNU Make, and optionally OpenGL, GLU, GLUT, and Doxygen. The build system uses GNU Automake, Libtool, and Autoconf (you usually do not need to install these, unless you want to use the development kit that includes a simulator and some other mobile robotics libraries).

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. Using interpolation to improve path planning: The Field D* algorithm. *Journal of Field Robotics*, 23(2):79–101, February 2006.

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

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

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

[6] Kurt Konolige. A gradient method for realtime robot control. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2000.

[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] Roland Philippsen. A light formulation of the E* interpolated path replanner. Technical report, Autonomous Systems Lab, Ecole Polytechnique Federale de Lausanne, 2006.

[12] 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.

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

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