

An Explicit Path Planner to Facilitate Reactive Control and Terrain Preferences

Robin R. Murphy
Center for Robotics
and Intelligent Systems
Colorado School of Mines
Golden, CO 80401-1887
rmurphy@mines.edu

Ken Hughes
Electrical and Computer
Engineering
University of the Pacific
Stockton, CA 95211
hughes@napa.eng.uop.edu

Eva Noll
Mathematical Sciences
Department
LaSalle University
Philadelphia, PA 19141
nolle1@lasalle.edu

Abstract

This paper presents a new approach to reactive control utilizing the output of an explicit path planner which considers terrain preferences. To demonstrate the concept, a wavefront propagation type of planner algorithm, Trulla, is integrated into a reactive framework and used as part of the overall architecture. Terrain preferences are represented as weights or costs which are considered in generating a near optimal path. The algorithm is demonstrated on the CSM Denning-Branch MRV4 mobile robot for three scenarios: navigation out of a box canyon, planning with terrain preferences, and opportunistic path improvement.

1 Introduction

Explicit path planning methods typically use *a priori* maps to generate a set of path segments for an autonomous mobile robot to follow. Important issues such as representation (e.g., configuration space, cell decomposition), optimality of the path (e.g., A* search), and speed of execution continue to be explored. However, two important issues affecting the practicality of these methods have been largely ignored.

First, explicit path planning methods are usually independent of the mechanism for actual navigation. For example, in some hybrid deliberative/reactive architectures [1], an explicit path planner uses *a priori* knowledge to generate a set of subgoals. The reactive portion of the architecture attempts to reach the first subgoal; once there, it then heads for the second subgoal, etc. As a result of this partitioning between path generation and reactive control, the robot may encounter unmodeled obstacles which make it (a) impossible to reach the subgoal and/or (b) miss opportunities to go directly to the next subgoal with significant navigational savings.

Second, existing path planning algorithms, with the notable exceptions of recent work by [2, 11], do not appear to factor in the costs of navigating over undesirable or uncertain terrain. Instead they attempt to represent the world as either “empty” or “occupied.” Note that some areas may be empty but not preferred.

For example, a user may wish the robot to stay on sidewalks whenever possible rather than go on grass or dirt. The grass and dirt have a lower preference associated with navigating through them. Also, it should be noted that in some cases the *a priori* map which serves as the foundation for path planning may not be completely known. Maps made from satellite data may not be able to identify with complete certainty whether an area is easily passable or not. Therefore, a plan planner should consider not only whether space is occupied or empty, but also the costs and/or risks associated with navigating through it.

This paper concentrates on the implementation of Trulla, an explicit path planner which incorporates terrain preferences, on the CSM Denning-Branch MRV4 mobile robot. It begins with a brief overview of the Trulla algorithm (Section 2) and the motivation for selecting it for this application. The implementation in C++ is presented in Section 3. Demonstrations using the CSM mobile robot are reported in Section 4, and the ramifications and related work are discussed in Section 5. Section 6 concludes that it is feasible for an explicit path planner to both bridge the gap between explicit path planning and reactive control and can handle preferences about terrains. Directions for future work are also noted.

2 Trulla

The Trulla algorithm is an optimal path planner which is suited for workspaces composed of regions weighted according to their cost of traversal. It shares similarities with path planning algorithms based upon wavefront propagations and potential fields. The terrain is represented by discretizing it into a m by n grid, and a goal location within the grid is identified as the starting point for the algorithm. Each grid location contains information regarding the weight of the region containing it, the direction of the path to the goal, distances to the next goal or subgoal along the path, and total distance to the goal. Also contained within each grid location is information regarding how the path to the goal last changed; for example, if the path to the goal must “bend” around an obstacle or region of high traversal cost, this information is stored and

propagated to other grid locations. When such path alteration occur, the grid location at which the change occurs becomes a subgoal for subsequent paths.

Since the final output of the algorithm is a discretized grid in which each grid location contains the direction and distance to the next subgoal, the robot can use this information to direct it toward the goal through a low-cost route. It also effectively directs the robot away from higher-weight regions where possible and always away from known obstacles within the terrain. The direction information can in essence be considered a type of potential field.

A thorough description of the algorithm can be found in [4]. The Trulla algorithm was originally developed as a simple path planning algorithm suitable for a hardware implementation, specifically to exploit parallelism and decision-making using only local information. These goals facilitated implementing the algorithm in a VLSI systolic array. Subsequent work in this area has yielded a version of the algorithm capable of such an implementation [7]. When run on a scalar processor, the Trulla algorithm is much less efficient.

The Trulla algorithm was selected as the candidate for inclusion in the reactive architecture for a number of reasons. Two principle motivations were (1) the grid-based representation used in finding the paths made it easier to choose a path when the robot's position is approximately known, and (2) the path representation used by Trulla for each path segment (essentially a vector whose direction points to a goal or subgoal and whose magnitude reflects the distance to the goal/subgoal) was compatible with the behavior-combining mechanism in place in our architecture. An additional feature of the algorithm is that it provides an optimal path to the goal location from *each* physically-accessible location in the grid. This can be particularly useful in a reactive system; avoidance behaviors may cause the robot to leave its original path but once the behavior returns control another optimal path can often be located without replanning.

We should briefly define the term "optimal path" when used in the context of a grid-based representation of a terrain. There is only one true minimum distance between any two points in the terrain (and not necessarily one path with this distance) and any discretization can only approximate this distance. The Trulla algorithm, like most algorithms using discretization, finds an optimal path with respect to the discretization whose distance is longer than the true minimum distance. The difference between these two distances increases as the grid coarseness increases. When the term "optimal path" is used in this paper, it refers to the shortest path within the constraints of the discretized grid.

Regardless of the reasons for which the Trulla algorithm was selected, many other path planning algorithms could have been selected in its stead. We will examine some of these algorithm and potential trade-offs in Section 5.

3 Implementation

The Trulla algorithm is embedded in the CSM hybrid deliberative/reactive architecture [5], and is written in C++. The layout of the code is shown in C++ pseudo-code below.

```
//
//deliberative component
//
LoadFile();           //a priori map
SetGoal(goalLocation);
GeneratePaths();
DisplayPaths();      //optional

//
//reactive execution component
//
while (IsGoal(currentLocation) != TRUE)
{
    subGoalLoc=NextSubgoal(currentLocation);
    move-to-goal(subGoalLoc);
    avoid-obstacle();
}
stop-robot();
```

Execution begins with the deliberative component, which asks for the *a priori map* and the goal location. The map is an ASCII file. It represents the workspace as a rectangular grid, each cell of which is assigned a weight corresponding to the difficulty in traversing the particular cell (or other cost). Valid weights range from 1 to 9, inclusive, and the letter *x*. The higher the weight, the higher the difficulty or cost of navigation. A weight of *x* indicates that an obstacle occupies that cell and traversal is impossible. These ASCII values are an implementation detail; the algorithm can work with any non-zero real-valued weights.

Next all paths to the destination are computed via the `GeneratePaths` function. These paths can be optionally displayed. Once the paths are known, the program then turns control over to the reactive execution component. The program requests the next subgoal given its current location. This subgoal is passed to a reactive **move-to-goal** behavior. The behavior sets the direction and velocity for the robot to start moving. **move-to-goal** runs concurrently with the **avoid-obstacle** behavior, so that the robot will veer if necessary to avoid collisions. As the robot moves, the `while` loop continues to update the current location of the robot, which in turn may change the next subgoal. **move-to-goal** will terminate inside the loop if the current location is the subgoal. If the subgoal was not the destination, **move-to-goal** will be reinstated with a new subgoal on the next iteration. Otherwise, it will stop execution when the robot reaches the goal (within some tolerance set in `IsGoal()`) and the `while` loop exits.

4 Demonstrations

The Trulla algorithm was implemented in C++ on *Clementine*, a Denning-Branch MRV4 mobile robot controlled by an onboard 66MHz 486 PC using MS-DOS. *Clementine* is shown in Figures 1, 2, and 3.



a.

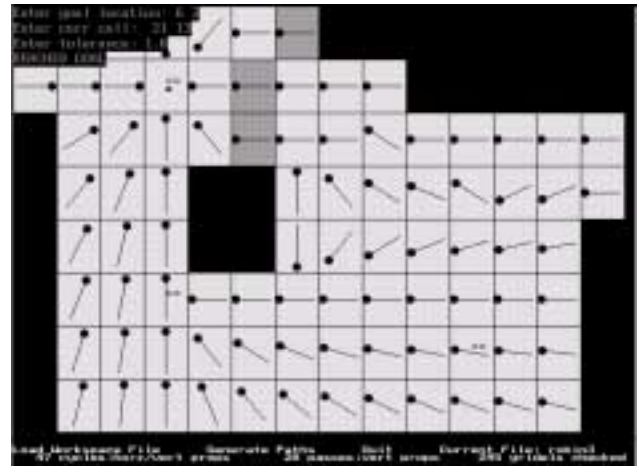


b.

Figure 1: a.) Trulla output and b.) Clementine navigating out of the box canyon.

Three scenarios are presented, each of which highlights some aspect of the planning algorithm and its integration into a hybrid deliberative/reactive architecture. In the box canyon scenario, the robot was trapped in a box canyon and needed an explicit plan to escape. This scenario demonstrated that Trulla is able to handle commonly cited cases as well as any other explicit path planner. The sensitivity of the plan to terrain preferences was demonstrated in the second scenario, where the robot chose a path over a cable on the floor only when there was no other cost effective way to go around it. In the third scenario, the robot plans a path and begins execution, then encounters a large obstacle. As the robot reactively avoids the obstacle, another path becomes optimal and the robot takes that route.

Each scenario was conducted in a 20 ft by 28 ft arena in the Mobile Robotics/Machine Perception Laboratory. The terrain was discretized for Trulla into two foot square grid locations. The floor was marked



a.



b.

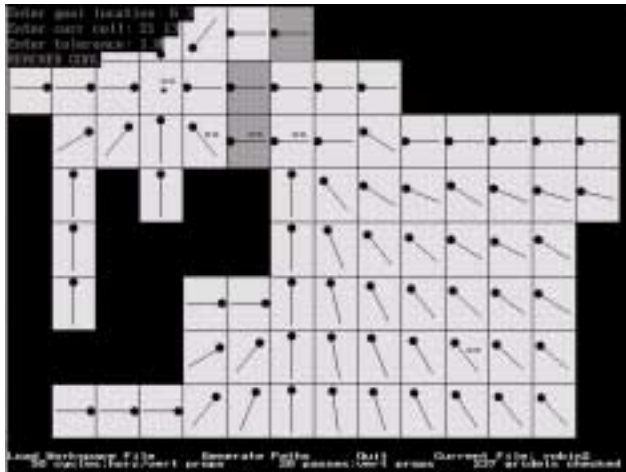
Figure 2: a.) Trulla output and b.) Clementine navigating around cord.

with yellow tape to allow the output of the graphics simulation to be easily matched to the test arena.

4.1 Box Canyon Scenario

The box canyon scenario is an example of why a robot may need to explicitly path plan. In the scenario, the robot becomes trapped in a dead end or blind alley. Although there are reactive behavioral methods that will eventually allow the robot to escape, the robot could generally exit more efficiently if it has created (or updated) a map and then plans the best route out. In this demonstration, the robot begins in a box canyon of paper mache "Mars rocks" and hay bales.

Trulla plans the set of paths shown in Fig. 1a. Each square is grid location representing a 2ft by 2ft area corresponding to a taped grid on the floor in Fig. 1b. The upper left corner is (0,0). The black squares on the graphics indicate the grid location is occupied, white squares are empty spaces. Arrows in the grid elements represent the approximate direction the robot



a.



b.

Figure 3: a.) Trulla output and b.) Clementine navigating over the cord.

should go if in the corresponding location; the round dot marks the head of the arrow.

The robot starts in the box canyon at (8,8) and has a goal of (4,19). The subgoals are shown as = superimposed over the arrows. In this case, there were seven subgoals (the start location counts as one). The robot backed out of the box canyon, turned clockwise and went outside of the box, then turned to follow the outer edge of the rocks. The robot using the Trulla path to exit the box canyon is shown in Fig. 1b.

4.2 Planning with Terrain Preferences

In order to illustrate Trulla's handling of terrain preferences, the robot was instructed to move between the same starting (21, 13) and goal (6, 3) locations for two different maps. The first map represents the power cord shown in Fig. 2b. as a partially passable terrain. The bundle of the power cord in the middle of the floor was marked as an obstacle (too high for the robot to roll over). The tail, running to the outlet on the wall on the right, was marked with

a value of 5, indicating that it was passable but that the robot should avoid the cord if possible. These are the three darker gray grid elements in Fig. 2a. It should be noted this type of traversability weighting scheme could be inferred from range or texture data. The second map (Fig. 3a.) shows the addition of an obstacle next to the power cord, preventing the robot from reaching the goal without running over the tail of the cord.

In the first case, Trulla computes a path which routes the robot around the cord. The path is only slightly longer than if it went over the cord, therefore on a cost basis, it is optimal (the robot going around the cord is shown in Fig. 2b.). In the second case, *Clementine* is forced to go over the cord, as seen in Fig. 3b.

Next, an obstacle (*C2*, the MR/MP laboratory's outdoor robot) is placed near the cord, blocking the robot's previous path. In this case, Trulla routes the robot over the cord because there is not other passable route. The Trulla output is shown in Fig. 3a.

4.3 Opportunistic Change to a Better Path

The third scenario illustrates how the robot can use the precomputed paths opportunistically. Fig. 4 shows a map with a large modeled obstacle in the center in black. An unmodeled obstacle (a partition) was placed in front of the robot. The short dashed line shows the actual path taken by *Clementine* to the goal using a purely reactive **move-to-goal** behavior. The long dashed line shows the actual path taken using Trulla. In both cases, the robot used an **avoid-obstacle** behavior, which attempted to reactively avoid obstacles by choosing the most open area nearest to the desired direction. In the non-Trulla case, the desired direction was always a straight line to the goal. As a result, the opportunity to make a hairpin turn was missed and *Clementine* took a longer route to the goal. In the Trulla case, as the avoid behavior drove the robot off the original course to avoid hitting the partition, the **move-to-goal** behavior was updated each execution cycle with the best precomputed direction to the goal (or subgoal). Therefore, it was able to take advantage of the opening afforded by the hairpin turn, as shown in Fig. 5, and take a shorter route to the goal.

The integration of path planning and reactivity is not seamless, however. We have encountered situations where the avoid behavior displaced the robot to a grid element where the precomputed path continued to direct the robot towards the unmodeled obstacle, resulting in an oscillating motion. This type of event ideally would trigger an update of the map and recomputation of paths.

5 Discussion

The above demonstrations illustrate that it is feasible to integrate an explicit path planner within a reactive control framework. They also show that some practical advantages can be gained depending on the choice of path planner, which in this implementation was the Trulla algorithm. This algorithm can provide plans according to terrain preferences and can

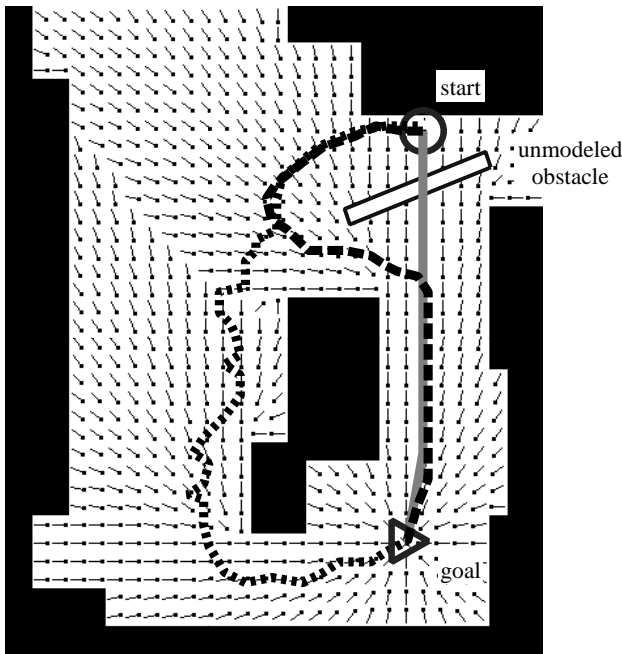


Figure 4: Layout showing unmodeled obstacle. Gray line shows original path computed by Trulla, long dashed line the actual path with Trulla, and short dashed line shows purely reactive path.

provide some alternative plans without the need for replanning. The implementation also proves to be relatively fast: path planning algorithms which rely on grid representations of workspaces tend to be computationally expensive, introduce digitization biases, etc. Even on a serial processor, Trulla is fast enough not to introduce any noticeable time delay for computation for indoor environments. Digitization biases and the resulting “jigsaw” like paths are not problems in Trulla because it implicitly relaxes the path segments similar to Thorpe [10]. A path can be drawn between any two grid locations, rather than being restricted to connecting neighboring locations via one of eight directions. The Trulla weighted grid representation is also consistent with the occupancy grid methods for avoiding obstacles. Therefore, Trulla could be readily adapted to use maps constructed by the robot as it explored an environment.

While the Trulla algorithm was chosen for integration with the reactive architecture in this instance it is by no means the only algorithm which could be used. Other path planner based upon wavefront propagation methods [6], potential fields methods, the D^* algorithm [8], and other philosophically similar systems [2, 11] could be modified and/or extended to fit the requirements of a reactive architecture. The degree of the integration’s success will depend on the strengths and weaknesses of each planner, particularly with regard to how the modifications or enhancements affect their operation. For example, the D^* algorithm is an optimal and efficient path planner which can re-

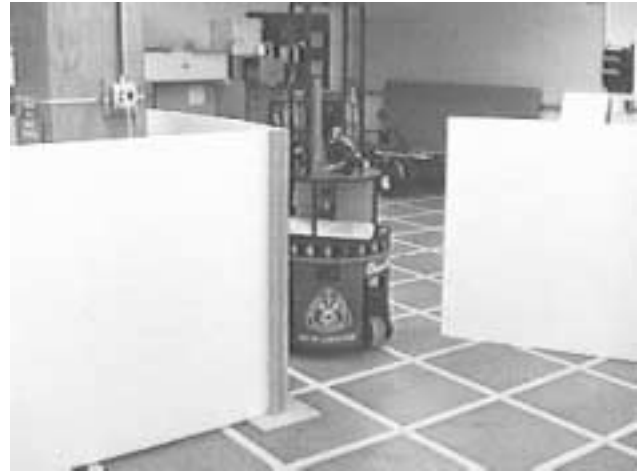


Figure 5: *Clementine* opportunistically making a hair-pin turn.

spond to dynamic changes in the environment and can function with incomplete information about the environment [9, 8]. It can operate on a discretized representation of a weighted terrain, using eight-connected neighbors, in a manner similar to Trulla. D^* could conceivably be integrated into the CSM architecture with little difficulty; however, since avoidance of unknown obstacles would cause the robot to deviate from the desired path, D^* would need to start replanning as soon as the reactive behavior is evoked if a “halt-and-replan” situation is to be avoided. This situation is avoided altogether in the opportunistic case for Trulla, but for other situations a complete replan must be computed since Trulla’s current implantation does not support partial replanning. Trulla does, however, have an advantage over D^* in its inherent relaxation of path segments versus paths strictly through the eight-connected neighbors.

The integration of Trulla into a truly autonomous robot is far from complete at this time. First, Trulla ignores how the weights of the grid locations are computed. While technically not part of the algorithm, it is significant. It is expected that in the future weights will be computed using more formal methods (e.g., cost functions [3]).

Another related issue concerns reactive execution: if the robot does not know where it is with respect to the map, it will not necessarily follow the optimal path. Localization may not be problem in execution for outdoors due to availability of GPS sensors, but it is a significant concern for indoor environments.

Finally, there is the question of when to replan in the face of unmodeled obstacles, terrain changes, etc. With the Trulla algorithm, one strategy would be to take advantage of its rapid computational speed to update the map and replan at the end of each sensing cycle. However, it is not difficult to imagine applications where the number of grid locations would drive the computational costs too high for this approach. Another strategy would replan only when a lack of

progress is indicated, such as an encounter with a box canyon. This would reduce computation but might also increase the length of time for the robot to reach a goal (i.e., it would waste time by not opportunistically improving its path). Instead, we are considering an approach similar to D* which would update the map and replan locally whenever some metric indicates a significant deviation from expectations (conceptually similar to but less demanding than [2]).

6 Conclusions and Future Work

The Trulla path planning algorithm is novel in that it integrates well with reactive control and considers terrain preferences. As a side effect of the propagation method, the near optimal path to the goal from all locations is computed. This allows the robot to react to unmodeled obstacles, veer off course to a location where the original path is no longer optimal, and then opportunistically resume optimal path execution with an alternative path. It further simplifies path execution because the robot does not necessarily have to reach a specific subgoal. If the subgoal is unobtainable (e.g., blocked), the robot may opportunistically use an alternative path as it reactively tries to approach the subgoal. The Trulla algorithm allows the cells in the *a priori* map to be weighted, indicating terrain preferences or costs as well as the location of obstacles. The algorithm then considers these weights in generating the set of paths. Another practical advantage is that the paths computed by Trulla have undergone implicit path relaxation. Furthermore, although Trulla was developed for use on a parallel processor, it is sufficiently fast that the implementation on a 66MHz 486 processor showed no noticeable time delay in computing the set of paths to the goal.

One important issue being investigated is the frequency and extent the grid should be updated and paths replanned. Another key problem actively being researched is localization in indoor environments: how does the robot know where it is with respect to the map?

The Trulla source code is available on request.

Acknowledgments

This research was supported in part by the Computing Research Association's Distributed Mentoring Program and the Colorado Space Grant Consortium. The authors would like to thank Dave Hershberger for integrating Trulla with the obstacle avoidance software and his assistance in the preparation of this paper.

References

- [1] Arkin, R. C., Riseman, E. M., and Hansen, A., "AuRA: An Architecture for Vision-Based Robot Navigation", *Proceedings DARPA Image Understanding Workshop*, Los Angeles, CA, February, 1987, pp. 417-413.
- [2] Gat, E., Slack, M., Miller, D.P., and Firby, R.J., "Path Planning and Execution Monitoring for a Planetary Rover," *IEEE International Conference on Robotics and Automation*, Cincinnati, OH, 1990, pp. 20-25.
- [3] Gifford, K.K., and Morgenthaler, G.W., "Optimal Path Coverage Strategies for Planetary Rover Vehicles," *46th International Astronautical Federation Congress*, Oslo, Norway, October, 1995.
- [4] Hughes, K., Tokuta, A., and Ranganathan, R., "trulla : an algorithm for path planning among weighted regions by localized propagations", *1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Raleigh, NC, July 7-10, 1992.
- [5] Murphy, R., and Mali, A., "Lessons Learned in Integrating Sensing into Autonomous Mobile Robot Architectures," in press, *Journal of Experimental and Theoretical Artificial Intelligence*.
- [6] Mitchell, J. S. B., and Papadimitriou, C. B., "The Weighted Region Problem: Finding Shortest Paths Through a Weighted Planar Subdivision", *Technical Report 885*, School of Operations Research and Industrial Engineering, Cornell University, 1990.
- [7] Ranganathan, R., Parthasarathy, B., and Hughes, K., "A Parallel Algorithm and Architecture for Robot Path Planning", *1994 International Parallel Processing Symposium*, Cancun, Mexico, April 26-29, 1994.
- [8] Stentz, A., "The Focussed D* Algorithm for Real-Time Replanning," proceedings *1995 International Joint Conference on Artificial Intelligence*, Montreal, CA, Aug. 20-25, 1995, pp. 1652-1659.
- [9] Stentz, A., "Optimal and Efficient Path Planning for Partially-Known Environments," proceedings *1994 International Conference on Robotics and Automation*, San Diego, CA, May 8-13, 1994.
- [10] Thorpe, C.E., "Path Relaxation: Path Planning for a Mobile Robot," *1984 International Conference on AI (AAAI-84)*, pp. 318-321.
- [11] Woodhead, R., Tucci, J., and Yao, J., "High Speed Multiple Vehicle Path Planning," FMC Corporation Technical Report, 1993.