The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

Variable Sized Grid Cells for Rapid Replanning in Dynamic Environments

Rachel Kirby, Reid Simmons, and Jodi Forlizzi

Abstract— **This paper presents a method for improving the runtime of an optimal heuristic path planner (A*) so that it can run repeatedly, in real-time, in a dynamic environment. This is necessary for mobile robots navigating in dynamic environments that have moving obstacles with associated costs, such as personal space around people or buffer zones around dangerous vehicles. Our approach is to modify the search space used by the A* algorithm, increasing the size of grid cells further from the robot. This approach relies on the notion that only the area closest to the robot needs to be searched carefully; areas further from the robot can be searched more coarsely. Because the planner is assumed to run repeatedly as the robot moves, the robot will always have a fine-grained path defined for its next action. We have experimentally verified in simulation that this algorithm can be run in real-time and produces paths that are comparable to full-resolution planning.**

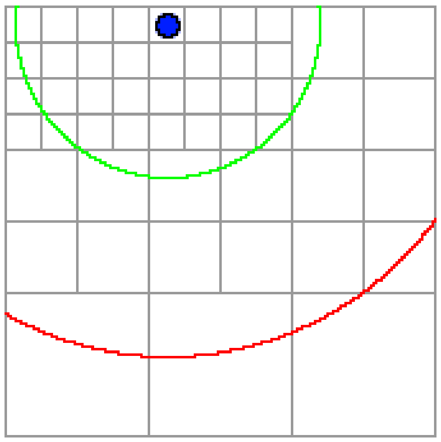

Fig. 1. A variable grid used for planning. The grid resolution decreases with the distance from the robot (blue circle). Shown are three grid sizes: the finest resolution is close to the robot (within the green circle), next largest is between the green and red circles, and the largest resolution is furthest away from the robot.

I. INTRODUCTION

Robots that operate in the real world need to respond rapidly to changes in the environment. A plan to the robot's goal, generated given available data, quickly becomes invalidated as the environment changes or the robot receives new information. A challenge in mobile robots, then, is replanning paths as quickly as possible. Especially challenging are environments with dynamic obstacles and obstacles with associated costs, such as personal space around people, buffer zones around dangerous vehicles, or rough terrain. Because sensors are imperfect, robots navigating in dynamic environments must replan whenever they receive new sensory data in order to ensure a safe, low-cost path.

Two main types of planners are currently used: heuristic search algorithms and randomized planners. Heuristic search algorithms, most notably A* [1], can find optimal paths, but typically do not run fast enough to replan in real time, as the robot receives new sensory data. Many variations on A* exist in order to improve replanning time, typically by saving and reusing portions of the search tree. Lifelong Planning A* (LPA*) [2] can rapidly replan when the environment changes, but only when planning from the same start state, and thus cannot be used for a moving robot. The replanning algorithms D* [3] and D* Lite [4] allow the start state to change, but do so by planning in reverse—from the goal state to the robot's current position. This works in many cases, but not with dynamic obstacles—the robot has no way of knowing where the obstacles will be at the time it reaches the goal, so the state of the world when the robot reaches the goal is unknown.

R. Kirby and R. Simmons are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA {rachelg, reids}@cs.cmu.edu

J. Forlizzi is with the Human–Computer Interaction Institute and the School of Design, Carnegie Mellon University, Pittsburgh, PA 15213, USA forlizzi@cs.cmu.edu

In contrast, Real-Time Adaptive A* (RTAA*) [5] and Generalized Adaptive A* (GAA*) [6] both plan forward, from start to goal, and allow for changing action costs. RTAA* handles only increasing costs, such as an obstacle perceived where the previous search assumed free space, and thus it is unable to handle dynamic obstacles. GAA* allows for action costs to increase or decrease; however, it is not designed to handle moving obstacles with associated costs.

A different approach to real-time replanning uses randomization, rather than exhaustive search. One common approach is to use Rapidly-Exploring Random Trees (RRTs) [7], [8], which are designed to explore the environment quickly. RRTs are guaranteed to find *some* path to the goal, but not necessarily an *optimal* path. Methods exist to heuristically bias RRTs to find the goal state more rapidly and partially account for path cost (e.g. [9]). However, despite biasing, RRTs do not find smooth or optimal paths. While the generated paths can be post-processed to yield smoother paths, doing so may eliminate legitimate avoidance maneuvers around moving obstacles.

Thus, none of these existing search algorithms completely account for real-time planning for a mobile robot in a dynamic environment where traveling near moving obstacles may have associated costs. Our approach, using a variable grid such as depicted in Figure 1, seeks to satisfy this need. In the remainder of this paper, we will discuss our approach and its implementation and will describe the tests we have run to verify the advantages of the method over existing search techniques.

II. APPROACH

Heuristic path planners rely on predictive heuristics, such as the remaining distance to the goal, in order to guide the search. Poor heuristics can cause the search to examine more nodes than necessary. Costs associated with obstacles (e.g. rough terrain)—and especially costs associated with dynamic obstacles (e.g. people)—are difficult for heuristic planners because these factors typically do not have useful predictive heuristics. Thus, when a heuristic planner encounters such an obstacle, it must expand a large number of nodes in order to find an optimal path. Unfortunately, heuristic planners such as A* typically have a run-time that is worse than linear in the number of nodes expanded. Reducing the number of nodes the search must expand thus improves the search time.

Our approach is to modify the search space used by the A* planner. In this way, the planner can be used unchanged. In particular, rather than performing the entire search on a regular grid, as most planning algorithms do, we decrease the resolution of the search further from the robot. That is, only the areas near the robot are searched carefully; areas further from the robot are searched more coarsely. Because this results in fewer search nodes, planning can occur rapidly. New plans can thus be generated repeatedly as the robot moves, so that the robot will always have a fine-grained path defined for its next action.

The most similar methods to our own are Quadtree and Framed-Quadtree planners [10], and other planners that quadtree-like hierarchical decompositions of space [11]. Quadtrees are irregularly-sized grids formed by recursively subdividing regions into four quadrants until each region is either free of obstacles or is the smallest allowed resolution. In sparse maps, quadtrees reduce the memory requirements (and thus search time) over regular grids. However, paths found with quadtrees are usually sub-optimal as compared to regular grids, particularly in large areas of free space. Framed Quadtrees create more optimal paths by modifying the quadtree data structure, but at the expense of greater memory requirements. In particular, framed quadtrees perform poorly when the world is generally known in advance.

In contrast, our approach uses a variable grid that is composed of regions of regular grids of decreasing resolution, spanning outward from the robot's position, as shown in Figure 1. The key idea behind this method is that, if the search can be done quickly enough, then the robot can regenerate plans at each timestep (as it gets new sensor information). Thus, the plan needs to be at a high resolution only near the robot; a rough path is sufficient further from the robot, because the robot will generate a new plan before reaching those areas. This approach is significantly different from the quadtree-based approaches in that the grid does not remain static between searches; rather, the grid changes as the robot moves, keeping the finest-resolution cells centered over the robot's position. In addition, by using an implicit representation of the changing grid cells, our approach does not require any additional memory over a typical A* search.

III. DESIGN CHALLENGES

Key design challenges in implementing the variable-grid-cell planner include: selecting the grid variations, representing dynamic obstacles, and handling the boundaries between resolutions.

A. Selecting the grid variations

One design consideration with this approach is how to select the grid variations: what resolutions to use, and at what distances to change the resolution. Close to the robot, the planner should use the finest resolution available (e.g. the map resolution). The distance at which the planner can switch to a coarser resolution is dependent primarily on the speed of the robot; the planner should always be able to provide a detailed path for several timesteps. Further away from the robot, a coarser resolution will yield faster path computation, as long as the grid cells are not allowed to be overly large for the environmental conditions.

An important consideration regarding this approach is that dense environments make path-planning on a larger grid cell size impossible. For example, suppose that the robot will need to navigate through a 1-meter-wide doorway close to the goal. If the grid cell size near the doorway is fairly large (say, 0.6m or 0.8m), then planning between cell centers may not find a free path through the doorway. This shortcoming can be avoided entirely if one has sufficient *a priori* knowledge of the environment and tailors the grid cell size accordingly. Unfortunately, this may not always be possible. In more complex environments, it may be necessary to perform sub-searches on some of the larger grid cells before declaring them impassable due to obstacles, perhaps using a method similar to the Framed Quadtree approach. Since our approach assumes that the robot has a map of the environment (which typically results in poor performance by Framed Quadtrees), one could pre-label obstacle-dense regions on the map that should always be searched at a fine resolution.

B. Dynamic obstacle representation

A common approach to planning with dynamic obstacles is to add time to the state space, effectively adding another dimension to the search space (e.g. [12]). Unfortunately, this adds a great deal more complexity to the search. Some approaches to managing this complexity include using random planners (e.g. [13]) or reducing the available action space (e.g. the "canonical trajectories" of [12]). In contrast, our approach is to include the dynamic obstacles, if present, in the state space. This yields several benefits over the state-time representation. To understand why, recall that the A* planner, with an admissible heuristic, does not need to examine a state more than once; if it encounters a previously examined state, the new path to that state is guaranteed to be

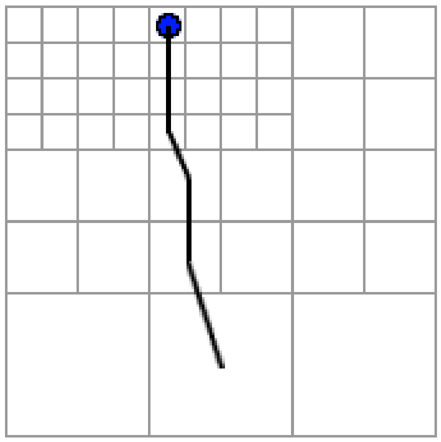

Fig. 2. A plan generated on the variable grid. Since plans are generated between node centers, a "straight" path may appear to have turns in it.

more costly than the one found initially. Adding time to the state space greatly decreases the chances of A* encountering the same state more than once, thus greatly increasing the search time. Adding dynamic obstacles to the state space results in more overlap. In addition, if the robot is allowed to ignore obstacles behind it or moving rapidly away from it (since such obstacles are unlikely to affect the robot's path), the state space can be simplified further. This allows the planner to consider dynamic obstacles with less complexity than a state-time space search entails, while also not limiting the robot's available actions.

Obstacles are assumed to move in continuous space, even though planning occurs on a grid. To account for this, two world states are considered the same if the robot is in the same state (position, orientation, and velocity) and if all dynamic obstacles are "close enough" to the same positions. We allow "close enough" to vary with the obstacles' distance from the robot, similar to the varying grid. Thus, when obstacles are far from the robot, their positions are considered more coarsely.

C. Resolution boundary challenges

The biggest implementation challenge in variable-grid planning is handling the boundaries between resolutions. Two problems arise: determining what actions occur at the boundaries, and aligning the search to the grid. These are illustrated by Figure 2. By assuming that the robot will always always generate a new plan before it reaches a section of the path that uses a larger grid cell size, both of these challenges can be overcome.

1) Actions at boundaries: Because actions late in the planned path are assumed never to be executed, the actions across resolution boundaries need only be approximate. Before the robot reaches any of the approximate actions on the plan, it will have generated a new plan with high-resolution initial actions. This can be guaranteed by simply having the navigation algorithm stop the robot if too much time has elapsed since generating the last plan. As we will show in Section IV, this is typically not necessary with our approach.

2) Grid alignment: Actions within each section of the grid should move the robot to a neighboring cell of the same resolution. As long as the coarser-resolution sections are integer multiples of the initial grid resolution, computing the within-section actions is trivial. However, at the boundaries between resolutions, actions will not necessarily move the robot to the center of the next grid cell. However, if we assume that the actions are only approximate, as mentioned previously, the path can be aligned to the grid by simply rounding the position to the center of the nearest grid cell (based on the distance from the start state). This allows planning to continue on a discrete grid.

Furthermore, we must consider obstacles within each grid cell. At the finest resolution available (near the robot), each cell is either occupied or free. However, at coarser resolutions, a grid cell may be only partially free, and partially filled with obstacles. One approach for handling obstacles would be to declare a cell that contains any obstacles to be completely blocked, but this may result in planner failure if even small obstacles appear as large blockades in coarser resolutions. Our approach is to check for obstacle collisions on the straight-line path between grid cell centers, using the Bresenham Line Algorithm [14].

Since we are able to align the robot's location to a grid cell of any resolution and we are able to compute each cell's occupancy on the fly, we are able to keep the variable grid implicitly defined over the given fine-grained map. That is, the variable grid incurs no additional memory requirements over the map itself.

IV. RESULTS

Experiments were run using the Carmen robot simulator[1]. The situation simulated is shown in Figure 3 and is that of a robot navigating through a hallway environment. The hallway is quite wide: approximately 4m across, narrowing to a 2m wide chokepoint. The cost function for the A* planner is a linear combination of multiple costs, including distance traveled, a preference for tending to the right side when passing people, and how much the robot encroaches into a person's personal space. That is, the planner attempts to find a path that minimizes travel distance while keeping the robot away from a person. The cost function is more fully described in [15]. Tests were run on an Intel Core2Quad processor running at 2.4GHz, under the Ubuntu Linux operating system. We simulate a 0.36m-diameter circular robot that has a Hokuyo URG scanning laser rangefinder, which receives data at approximately 10Hz.

At each state, the robot has three potential actions: move straight forward, turn $45°$ and move diagonally left, or turn $-45°$ and move diagonally right, each constant speed (0.5 m/s). We compare the results of a standard A* planner along with our variable-cell planner in two scenarios, as follows.

[1]Carmen software is available online at `http://carmen.sourceforge.net/home.html`.

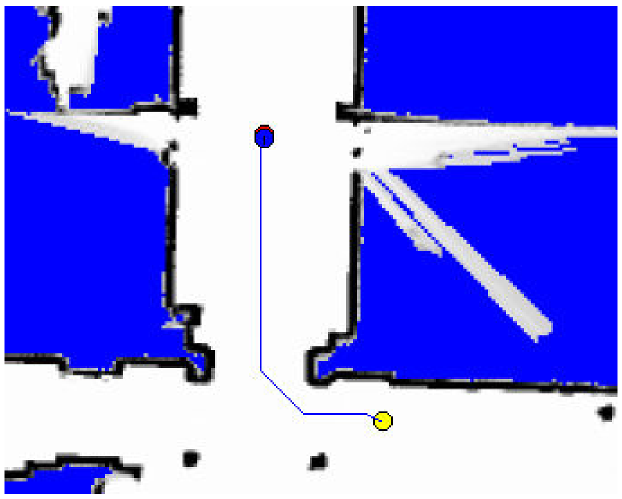
Fig. 3. Path found on a constant grid cell size of 0.2m. Robot is in blue, and the goal is in yellow. Search required 1547 node expansions.

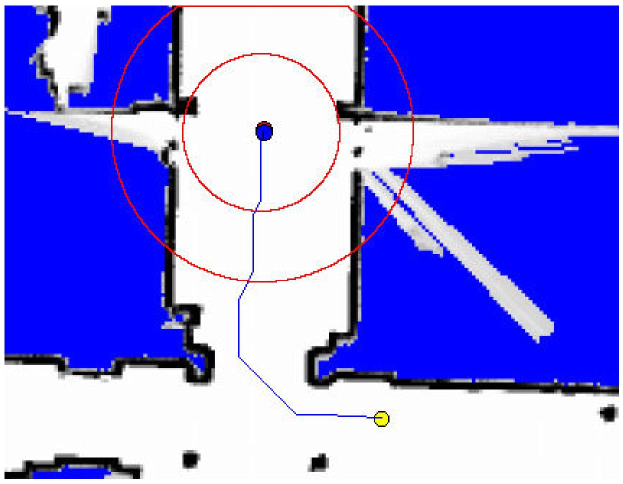
Fig. 4. Path found with a varying grid cell size. Red circles indicate distances at which cell size increased (2m and 4m from the robot). Search required 555 node expansions.

A. *Simple case: no moving obstacles*

The first test required the robot to plan a path from start to goal on a known map with no additional obstacles, as shown in Figure 3. The shortest path is approximately 10m long, and required the robot to make a left turn down a hallway. For the base case, the planner searched on a static grid with 0.2m cells. This search expanded 1547 nodes, and took an average of 0.050 seconds per search (over 10 trials).

Using our approach, we increased the grid cell size at 2m and 4m from the robot. In particular, the cell size was 0.2m from the robot to 2m out (as in the base case), doubled to 0.4m between 2 and 4m from the robot, and doubled again to 0.8m beyond 4m. These regions are indicated by circular outlines in Figure 4. The robot traveled at 0.5 meters per second, so these resolution changes allowed for four seconds of travel between each section.

In this case, the search required only 555 node expansions, and took an average of 0.043 seconds per search. While the variable-grid search expanded only about one third the number of nodes used in the static-grid search, the search time was similar. This is due to the overhead involved in initializing the search.

The paths generated by both methods are similar, though not identical due to the different grid resolutions. Figure 4 clearly demonstrates the changes in the path due to the varying grid. However, because our method is intended for use in rapid replanning, of note is that the first action generated by both methods is the same. As the robot moves and replans, the finest resolution of the grid also moves, continually producing optimal first actions.

B. *Dynamic obstacles: single case*

The second test case, shown in Figure 6, included a person moving directly toward the robot's initial position. In this model, a person is assumed to have a "personal space" buffer, which takes the shape shown in Figure 5. The cost is based on the relative velocity between the person and the robot,

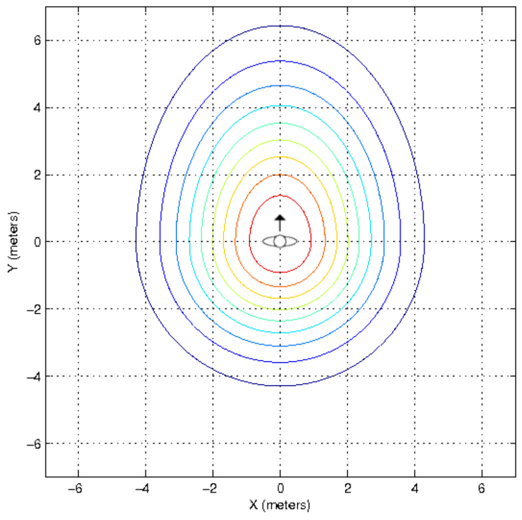

Fig. 5. Personal space cost function for a person moving along the Y-axis, with a relative velocity of 1m/s with respect to the robot. Greatest cost is centered over the person at point (0,0). Cost is computed by composing two Gaussian functions.

and the shape of the cost function is based on empirical findings from human studies literature (e.g. [16], [17]). The costs move with the person, requiring the robot to predict the person's path; currently the person is assumed to travel in a constant direction.

On a constant grid, the search expanded 29894 nodes and took an average of 0.73 seconds per search (averaged over 10 runs). Obviously, this speed is sub-optimal for a robot moving in real-time—in our example, the robot would have traveled about 40cm before reacting to any unexpected changes in the world.

As in the previous experiment, we performed the same search using varying grid cell sizes of 0.2m, 0.4m, and 0.8m. This setup is shown in Figure 7. This search expanded only 11724 nodes, taking an average of 0.26 seconds per search. Note that this search required only about one third the search

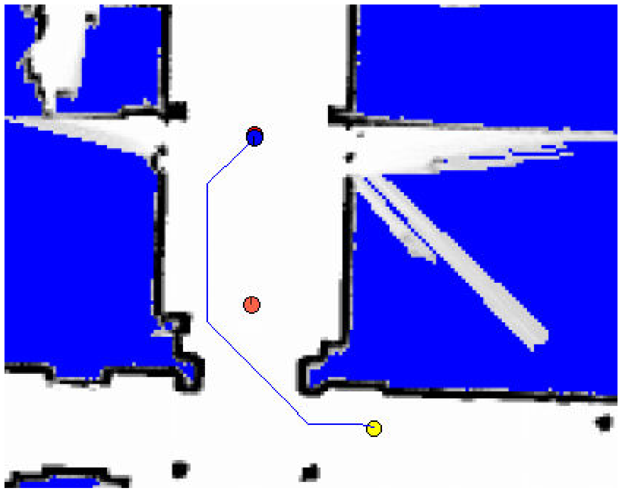

Fig. 6. Path found on a constant grid cell size of 0.2m. Robot is in blue, goal is in yellow, and a person (with personal space cost) is in orange. The person is moving toward the top of the image. Search required 29894 node expansions.

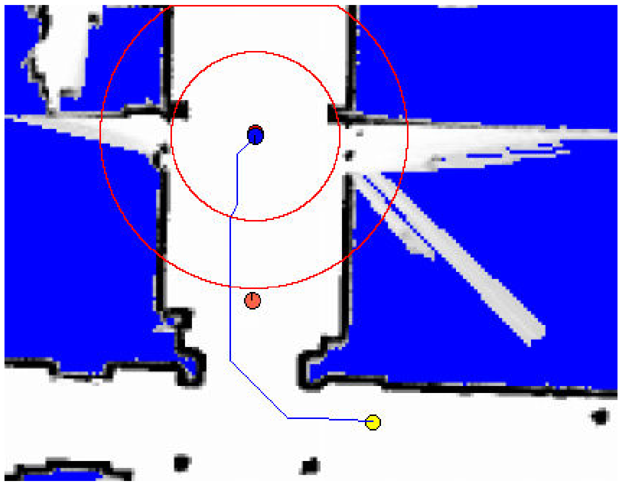

Fig. 7. Path found with a varying grid cell size. Red circles indicate distances at which cell size increased. Search required 11724 node expansions.

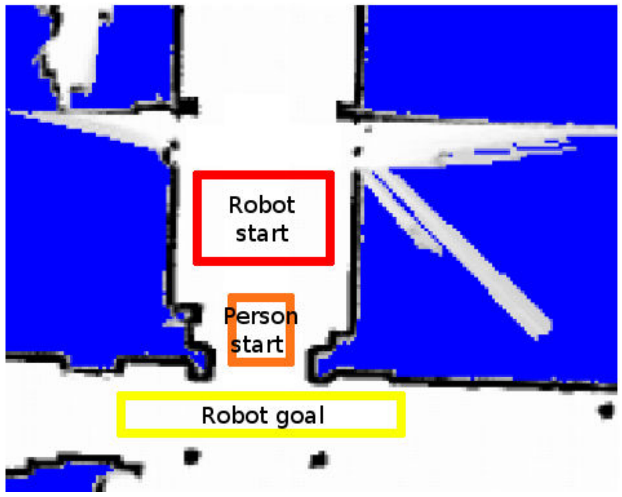

Fig. 8. Regions from which scenarios were drawn. The robot always began somewhere in the "Robot start" region, facing downward. The person began with a random orientation in the "Person start" region.

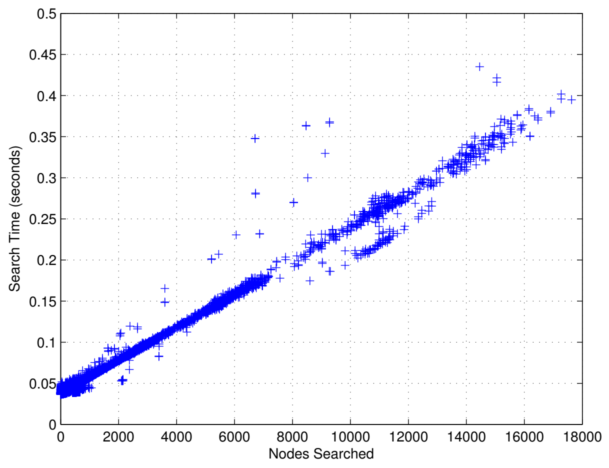

Fig. 9. Search times versus number of nodes searched, representing one hundred complete runs (including planning and execution cycles) of the scenario shown in Figure 7.

time as the pure A* search. Furthermore, the planner is able to run fast enough to replan 4-5 times per second.

As before, the two methods produce similar but not identical paths; again, the first action is the same in both approaches.

C. Dynamic obstacles: repeated tests

To determine whether the variable-grid planner would produce the optimal first action in all cases, we ran a set of 100 scenarios. Each scenario was similar to the ones given above, but with random start and goal locations for the robot and a single person with a random start location and orientation. The various locations were constrained to certain sub-regions of the map, as shown in Figure 8. Each scenario was run using A* on a static grid and on the variable grid set up as described above.

The variable-grid approach searched significantly fewer nodes than the static-grid method (average ratio of 0.33, $p < 0.01$ in a one-way ANOVA) and took significantly less time (average ratio of 0.63, $p < 0.01$ in a one-way ANOVA). In 87% of the trials, the first action found using both methods was identical. In all cases, the varying-grid approach took less time than the identical static-grid search.

D. Navigation

Finally, we ran several complete simulations with the robot planning and navigating to the goal, as in the scenario shown in Figure 7. That is, the robot navigated to a goal approximately 10m away, around a corner, while a person traveled directly toward the robot. The simulator added noise to the person's traverse, so that each run of the simulation was not identical. The robot generated new plans as quickly as possible, while navigating at a default speed of 0.5m/s. The robot followed the generated plans using the Pure Pursuit path-following algorithm [18], which guides the robot back onto the path if it strays or if a new path is planned. Each plan was followed until a new plan was received. Figure 9 shows the results of one hundred complete path traversals, composed of 20,106 planning cycles (approximately 200 plans generated per traversal). The longest-running search took 0.43 seconds, expanding 14,459 nodes. However, 92.8% of the planning cycles were completed in less than 0.1 seconds (searching less than 3500 nodes), which corresponds to the data rate of the laser sensor. Even with occasional outliers of half-second searches, our planning method is capable of reacting to real-time changes in the environment.

V. CONCLUSIONS AND FUTURE WORKS

In this paper, we have presented a straightforward method for improving the search speed of an A* planner. Our method runs quickly enough to allow the algorithm to perform repeated searches in real time, which we have verified in simulation.

Our research focuses on situations where a robot is moving toward a goal amidst moving obstacles and where the obstacles may cause changes to the cost function (e.g. personal space around a human). Typical replanning algorithms are unable to handle these moving obstacles and complex cost functions. Our approach is instead to replan from scratch with each iteration of the planner, but by modifying the search space, each search can occur rapidly. Rather than use a regular grid, our approach is to increase the grid cell size outward from the robot's start state. This results in a coarse path close to the goal, but a fine path near the robot. Since this allows the planner to re-run rapidly, the coarse sections of the path will never be executed. In this way, the planner can keep up with a changing environment.

We have validated our approach in several simulated scenarios, including planning and navigation cycles over a 10 meter traversal. We are currently beginning testing on a physical robot operating in the real world.

Though our approach is straightforward, we feel that it is a powerful solution to the complex problem of navigation in dynamic environments.

ACKNOWLEDGMENT

This research was partly funded by National Science Foundation Grant IIS-0624275.

REFERENCES

[1] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths in graphs," *IEEE Transactions on Systems Science and Cybernetics*, vol. SSC-4, no. 2, pp. 100–107, July 1968.

[2] S. Koenig, M. Likhachev, Y. Liu, and D. Furcy, "Incremental heuristic search in artificial intelligence," *Artificial Intelligence Magazine*, vol. 25, pp. 99–112, 2004.

[3] A. Stentz, "The D* algorithm for real-time planning of optimal traverses," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-94-37, Sept. 1994.

[4] S. Koenig and M. Likhachev, "D* lite," in *Proceedings of the AAAI Conference of Artificial Intelligence (AAAI)*, 2002, pp. 476–483.

[5] ——, "Real-time adaptive A*," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2006, pp. 281–288.

[6] X. Sun, S. Koenig, and W. Yeoh, "Generalized adaptive A*," in *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2008, pp. 469–476.

[7] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Computer Science Department, Iowa State University, Tech. Rep. TR 98-11, Oct. 1998.

[8] S. M. LaValle and J. J. Kuffner, Jr., "Randomized kinodynamic planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1999, pp. 473–479.

[9] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," in *Proceedings of IEEE International Conference on Interlligent Robots and Systems (IROS)*, Las Vegas, Nevada, Oct. 2003, pp. 1178–1183.

[10] A. Yahja, A. Stentz, S. Singh, and B. L. Brumitt, "Framed-quadtree path planning for mobile robots operating in sparse environments," in *Proc. IEEE Intnl Conf on Robotics and Automation, (ICRA)*, May 1998, pp. 650–655.

[11] K. Fujimura and H. Samet, "A hierarchical strategy for path planning among moving obstacles," *IEEE Transactions on Robotics and Automation*, vol. 5, no. 1, pp. 61–69, Feb. 1989.

[12] T. Fraichard, "Trajectory planning in a dynamic workspace: a 'state-time space' approach," *Advanced Robotics*, vol. 13, no. 1, pp. 75–94, 1999.

[13] M. Zucker, J. Kuffner, and M. Branicky, "Multipartite RRTs for rapid replanning in dynamic environments," in *Proceedings of the IEEE Int. Conference on Robotics and Automation (ICRA)*, Apr. 2007.

[14] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, pp. 25–30, Jan. 1965.

[15] R. Kirby, R. Simmons, and J. Forlizzi, "COMPANION: A constraint-optimizing method for person–acceptable navigation," 2009, to appear in the Proceedings of the IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN).

[16] E. T. Hall, *The Hidden Dimension*. New York: Doubleday, 1966.

[17] N. L. Ashton and M. E. Shaw, "Empirical investigations of a reconceptualized personal space," *Bulletin of the Psychonomic Society*, vol. 15, no. 5, pp. 309–312, 1980.

[18] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-92-01, Jan. 1992.