Cooperative Multi-Robot Path Planning

E. Colon, K. Verbiest

Abstract

In risky applications it is advantageous to use multiple robots in order to improve performances. In a research or patrol scenario robots have to move cooperatively in order to avoid collisions and to improve performance. In this paper we describe the implementation of two variations of a cooperative planning algorithm: Cooperative A* and Cooperative Voronoi A*. The task is decoupled into a series of single agent searches. The individual searches are performed in a 3D space-time search space and take into account the planned routes of other agents. The method is able to plan in 2D and 2.5D environments by incorporating traversability information. The algorithm can also handle single and multiple waypoints. These can be chosen by the user or automatically generated in function of an a priori known map and the characteristics of the detection sensors. A simulator has been developed in order to rapidly evaluate and compare algorithms and to analyse the influence of configuration parameters. A summary of the results and their discussion are presented in this paper.

Keywords: multi-robot, cooperative robotics, path planning

Introduction

In certain situations, such as surveillance, transport…, the use of multiple robots can be beneficial. The robots are required to accomplish their task autonomously without colliding with each other or the environment, preferably taking the shortest possible paths.

Generally two different strategies are used to control mobile robots: a *behaviour-based* approach and a *planning-based* approach. In this report, focus will be on global planning, and the planning based approach will be used. A plan will be developed based on a model of the environment and the robot, which will navigate the robots without collision through the environment and accomplish specified tasks. This can be combined with the reactive-based approach to create a typical mobile robot application: the global path plan can be paired with a reactive local navigator to circumvent small obstacles or react to unexpected circumstances.

Depending on the application different strategies exist to tackle this problem. Some popular algorithms for multi-robot planning use a decoupled approach and manage the complexity of the problem by planning trajectories for robots individually and sequentially. Other approaches use a randomized algorithm, such as probabilistic roadmaps (PRM), or follow the design of path-velocity decomposition. Such methods are however not guaranteed to find a solution if one exists. The planning algorithms Cooperative A* and Cooperative Voronoi A* described further, follow the decoupled approach.

Path Planner Simulator

The simulator presented here is implemented in C# on the Windows platform. It provides a tool for visualization and allows to test the capabilities offered by the path planning algorithms. Figures 1, 2 and 3 give a representation of the graphical user interface.

The path planner GUI is composed out of two main components:

- A menu bar allowing for user input. The users can choose the map and one of the available search algorithms. Several configurations settings can be set. By clicking on the map, the user can set the start, goal and optional waypoints for each robot. Several visualization options are possible.
- A visualization area. In this area the map, the robots, their start, goal and waypoints will be shown. An animation of the robots moving on their generated paths will illustrate more clearly how collisions among robots are avoided.

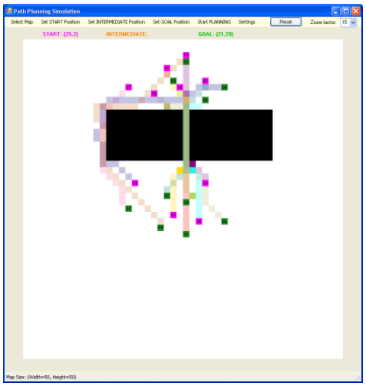

Figure 1: Screenshot of the path planner simulator. The paths are generated with the Cooperative A* algorithm.

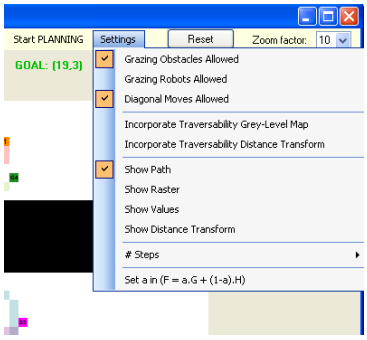

Figure 2: View of the settings menu.

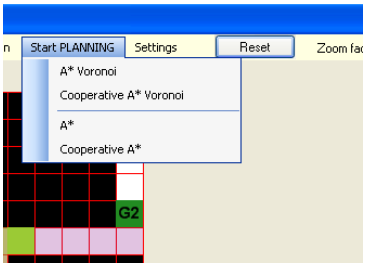

Figure 3: View of the available search algorithms in the planning menu.

Map format

A map is used to model the environment and to localize the objects of interest in the environment and for planning the route of robots by finding obstacle-free paths between the current position and a goal position. Different representations exist that are suitable to solve different aspects of the planning problem. We can distinguish two main categories of representations: roadmaps (visibility graphs, Voronoi diagrams, Probabilistic Roadmaps (PRM)…) and geometrical maps. Geometrical maps, more specifically, approximate cell decomposition maps were chosen for this simulator.

For either 2D or 3D environments it is possible to turn a bounded portion into discrete rectangular cells (2D array of cells) that may or not be occupied. The resolution of this discretization determines the number of cells per axis and the quality of the approximation. The representation may be considered as a binary image in which each *1*-state corresponds to a region containing at least one point of an obstacle and where *0*-state represents those regions that are completely free of obstacle points. The free space is defined as the region where robots can freely move. The obstacles are regions of space that are impassable for the robots. For each coordinate of this array a single bit has to be stored to indicate the state of the space. Such a map is a good choice when a mobile robot builds or updates a representation of its environment with its sensors. This 2D view, although a simplification, is generally is generally sufficient to plan robot motion paths in flat indoor or nearly flat outdoor environments. Figure 4 shows an example of a 2D binary map.

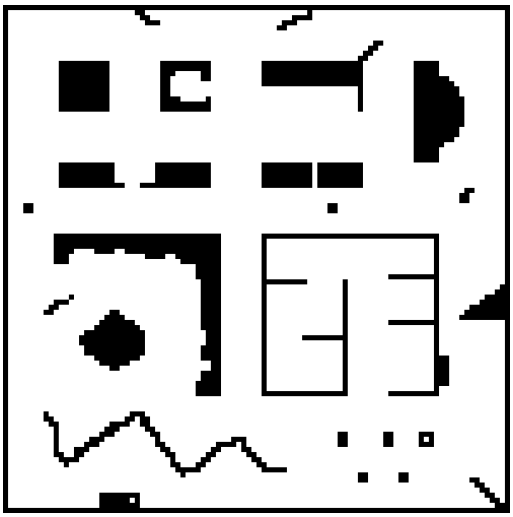

Figure 4: Binary 2D-map with equal traversability in the free space. Free space is visualized by white pixels (binary value 0, pixel value 255). Black represents the obstacles (binary value 1, pixel value 0).

Figure 5: Grey-level map. The darker regions correspond to a higher traversability penalty.

2.5D maps are a elaboration on 2D binary maps and are visualized as a gray-scale map, with a numerical value assigned to each cell. For instance, elevation maps give for each coordinate a corresponding height *h*. This information can be used in indoor environments but is most useful in terrain maps for outdoor navigation. Knowing the height allows to compute the slope in order to avoid too steep trajectories. Additional information can be further associated with each cell, introducing quantities like terrain roughness, communication, obstacle presence probability. An example of a 2.5D map is shown in figure 5. Both the 2D and the 2.5D maps are loaded as bitmaps into the simulator.

Configuration Possibilities

Depending on the planning problem different factors may be of interest. In the simulator different settings are possible which will lead to different paths being generated. A summary of the configuration settings is given below. A more in depth description of some will be discussed in following sections.

- **Traversability:** This setting gives the option for incorporating the grey-level values in a "height"-map. The cost to traverse a cell will then be dependant on the grey-value. This value can represent a number of characteristics such as: height, terrain traversability, safety …or a combination of these.
- **Trade-off speed/optimality:** The weight of the heuristic value can be adjusted to influence how the A* algorithm will generate paths. According to the weight of the heuristic value, the algorithm will be between two extremes: Dijkstra's algorithm and the Best First Search (BFS) algorithm.
- **Diagonal moves:** This setting can be set to either allow that robots move diagonally or not.
- **Distance transform:** This setting can be set to incorporate the distance transform information when generating paths. The distance transform map will then be used as input map in much the same way as a grey-level "height" map, leading to paths laying further from static obstacles in the scene.
- **Grazing other robots:** This can be set to either allow robots to graze each other or not, while moving diagonally.
- **Grazing obstacles:** This setting can be set to either allow robots to graze obstacles or not, while moving diagonally.

Path Scoring A*- Trade-off Speed/Optimality

Once a representation has been chosen to model the environment, a search algorithm is used to find the best path through this representation. The A* search algorithm has been used frequently in robotics to find the shortest path for a robot in a graph-based map, and has been proven to be complete and optimal.
The key to determining which cells to explore when figuring out the path is given by:

$$F(n) = G(n) + H(n)$$

Where:

- G (n) = the movement cost to move from the starting position (S) to a given cell n on the map, following the path generated to get there.
- H (n) = the heuristic estimated movement cost to move from that given cell n on the map to the goal position (G).

The Heuristic can be used to control the algorithm's behaviour. At one extreme, if H(n) is 0, only G(n) will play a role, and A* will turn into Dijkstra's algorithm, which is guaranteed to find the shortest path. At the other extreme, when G(n) is 0, only H(n) will play a role, and A* will turn into The Best First Search (BFS) algorithm. H(n) must be an underestimate for the real movement cost in order for A* to be able to find the optimal path. Different distance measurements can be used for both G(n) and H(n), for instance: Manhattan Distance, Euclidean Distance, Chessboard Distance,… In the simulator the cost for a diagonal move will be $\sqrt{2}$ times the cost for a horizontal or vertical move.

Depending on the need between speed or optimality the role of the heuristic value can be adjusted. In some situations it is better to have a good path quickly as opposed to a computationally expensive perfect path, especially in multi-robot environments and when the environment is not perfectly known. This trade-off between speed and optimality is built in as a configuration parameter in the simulator.

Cooperative Path Planning

Cooperative path planning is a multi-robot path planning problem where robots must find non-colliding routes to separate destinations, given full information about the routes of other robots.
Cooperative A*(CA*) [1] is an algorithm for solving the cooperative path planning problem. The task is decoupled into a series of single robot searches. The individual searches are performed in 3D space-time, and take account of the planned routes of other robots. A wait move is included in the robot's action set, enabling it to remain stationary. After each robot's route is calculated, the states along the route are marked into a reservation table. Entries in the reservation table are considered impassable and are avoided during path searches by subsequent robots. The reservation table represents the robots' shared knowledge about each other's planned routes.
The reservation table is implemented as a 3D grid (two spatial dimensions and one time dimension). Each cell of the grid that is intersected by the robot's planned route is marked as impassable for precisely the duration of the intersection, thus preventing any other robot from planning a colliding route.
It is important to note that any decoupled, greedy algorithm that precalculates the optimal path will not be able to solve certain classes of problem. This can happen when a greedy solution for one robot prevents any solution for another robot, for example see Figure 6. In general, such algorithms are sensitive to the ordering of the robots, requiring sensible priorities to be selected for good performance.

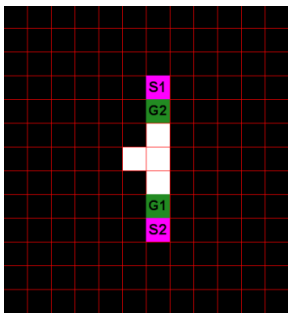

Figure 6: The solution for one robot prevents the solution for the other one. This is a consequence of the decoupled greedy algorithm.

In this simulator it is assumed that the robots are all identical and thus move at the same speed. Either they move at this speed, or they stand still. Furthermore, they are modelled as points (the size of one cell) in the map. Obstacles can simply be extended to take into account the size and the motion constraints of the robots.
Some results of paths generated by this algorithm can be seen in figures 7, 8, 9 and 10. The start and goal positions for the different robots are denoted by S1, S2, S3… (magenta) and G1, G2, G3… (green) respectively. Optional waypoints (intermediate points) are denoted by I1, I2, I3… (orange). Different robot paths are shown in different colours.

Traversability

In case of a 2.5D map obstacles are represented as occupied cells and traversability (cost of traversing a cell) of the free cells is indicated with a grey-value, ranging from a completely free (white, pixel value 255) to an occupied cell (black, pixel value 0). The higher the cost at a grid position the greater the penalty to traverse this position. A linear relation is used to determine the penalty from the value at a grid position:

$$traversal\ cost = \alpha.(\frac{255 - greyValue}{255})$$

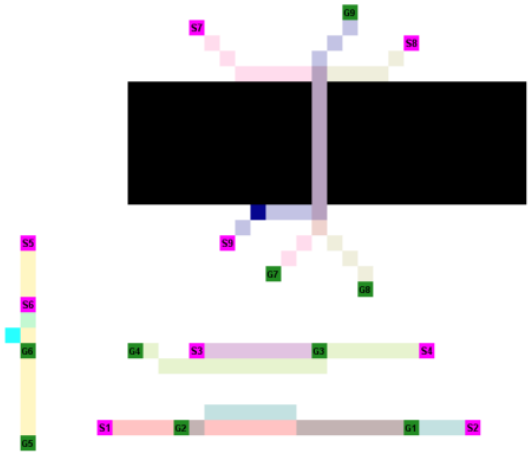

Figure 7: Typical results of path planning with the cooperative A* algorithm.

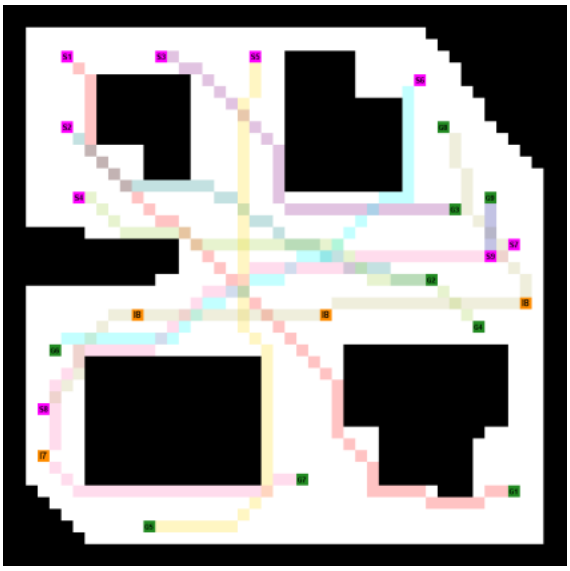

Figure 8: Typical results of path planning with the cooperative A* algorithm.

Figure 9: A map with small passages. Cooperative A* was used to generate the paths for the robots. This map has coves where robots can wait to let other robots pass to avoid collisions.

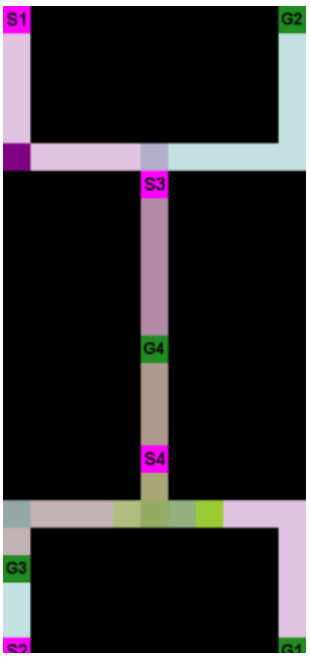

Figure 10: Maps with small passages are challenging. When one robot path doesn't prevent other paths to be found, solutions still can be generated as shown.

With α, a parameter which will determine the weight the grey-values will have in the total cost function. The higher this value the more the grey-values will be avoided, and safer paths will be generated. To the other extreme, when this value is set to 0, grey-values will be totally ignored, and paths will be generated as if there were no grey- values.
The traversability value can entail not just terrain traversability, but also other characteristics, such as time exposed to danger, visibility, energy expended… Consequently, applying the A* algorithm to these maps does not necessarily produce the shortest path, but the path with the lowest cost. A general mapping from these characteristics to the grey value is:

$$greyValue = w_1.characteristic_1 + w_2.characteristic_2 + \cdots + w_n.characteristic_n \quad \text{with}$$

$$w_1 + w_2 + w_3 + \cdots + w_n = 1$$

The characteristics in this formulation range from 0 to 255, just like the resulting grey value. The higher a specific characteristic value, the worse the traversability is coming from this characteristic. Different weights $w_1, w_2, w_3, \ldots, w_n$ can be used to alter the weight of the different characteristics in the final grey value.

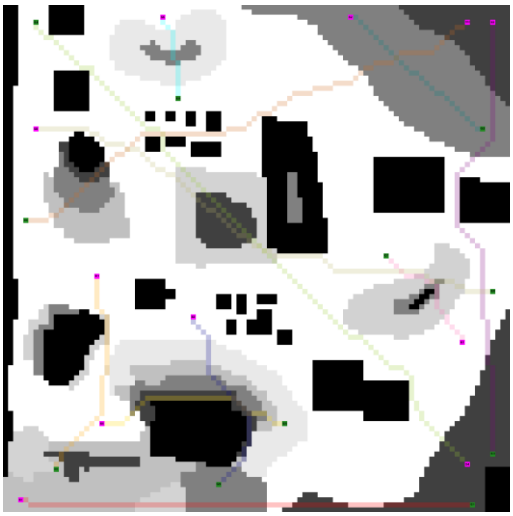

Figure 11: Path planning in a grey-level map. When the grey-levels are ignored the paths will be generated as if all open space has the same traversability. Only the obstacles will have to be avoided.

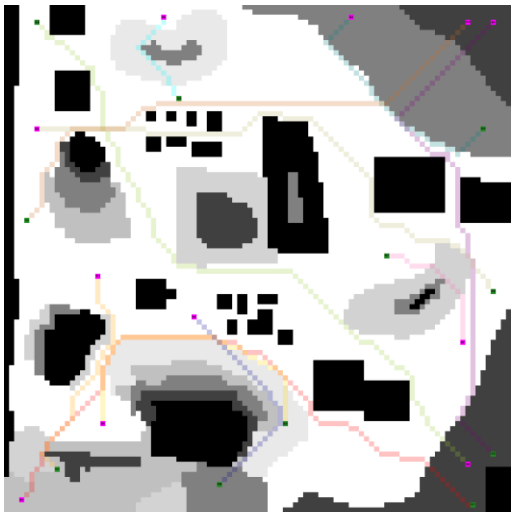

Figure 12: Path planning in a grey-level map. The traversability of the grey-levels is taken into account while generating the paths. This leads to different paths compared to figure 11 as the traversability of the terrain is incorporated.

In figures 11 and 12 the same grey-level map is shown. For the same problem set-up, the paths are calculated, once without incorporating this traversability information in figure 11, and once with this traversibility information taken into account in figure 12. As can be seen when comparing the results, different paths are generated. The paths generated in figure 12 will avoid terrain with difficult traversability if this leads to lower cost paths. But terrain with a low traversability is still accessible. Moreover, if the cost to go around the region with low traversability will lead to higher cost paths, the robot will go right through.

Distance Transform

Calculating the distance transform for a map can be useful for calculating robot paths that will be at a safer distance from obstacles in the scene. Different metrics are possible for calculating the distance transform, for instance the Manhattan distance, the Euclidian distance, the Chessboard Distance… The underlying metric used in this simulator is the Chessboard Distance. In figure 13 and 15 a map and its distance transform are shown.
When the distance transform map is treated as a grey-level map for path planning, with the grey-level values representing the difficulty for traversing that specific position, this will lead to safer paths.
In figures 13 and 14 this difference is apparent. In figure 13, the paths are calculated with the original map and in figure 14 with the distance transformed map. As can be seen from figure 14, when

incorporating this information, this leads to paths that are further removed from the obstacles. While searching for the paths for the different robots the lighter regions will be preferred, as the darker regions will have a greater penalty for traversing.

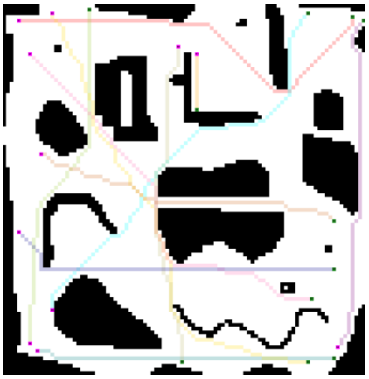

Figure 13: Binary map with equal traversability in the free space region. The paths are calculated without taking into account the distance transform information.

Figure 14: The paths in this map are calculated while incorporating the information available from the distance transform of the map, resulting in paths further away from the obstacles.

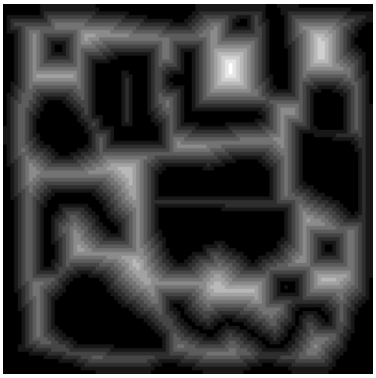

Figure 15: This is the map displayed in figure 13, after a distance transform is applied. Consequently the paths with best traversability will be at equal distances from the obstacles.

Cooperative Voronoi A* Path Planning

From the occupancy grid it is possible to calculate the Voronoi map, consisting out of Voronoi *edges* and *vertices*. When a robot follows a path on a Voronoi edge, it will be equidistant from all the points in the obstacle region. The vertices are the points where the edges meet. This leads to the generation of safe paths that pass with maximal clearance around the obstacles. In figures 16 and 17 a map and its Voronoi map are shown.

When combining the Voronoi diagrams with the previous Cooperative A* algorithm, the search-space will be reduced, leading to paths generated further from the obstacles and faster computation.

The start, goal and optional waypoints that the robot must visit are supplied by the user. When the starting, waypoint and goal positions of the robot are not on the Voronoi map, the points closest to them on the Voronoi map are calculated. These points are then connected to the Voronoi diagram via the Cooperative A* algorithm, as shown in figure 18.

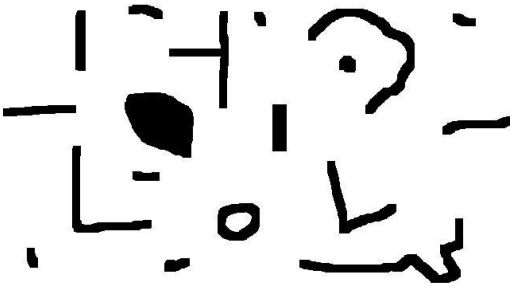

Figure 16: A map containing several obstacles.

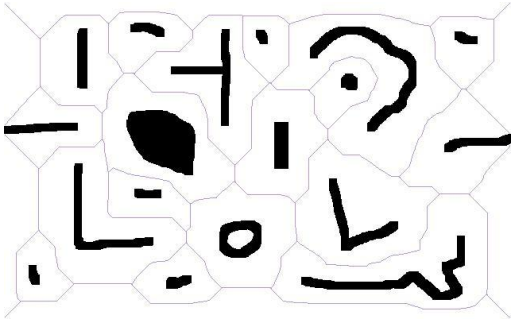

Figure 17: This is the map displayed in figure 16 augmented with its Voronoi map

To avoid collisions amongst the robots multiple Voronoi 'lanes' are used instead of just one. In figure 19, there are three lanes. The middle one has preference over the outer ones. This is accomplished by assigning an increasing traversability penalty for the outer lanes.

Figure 19 shows an example of paths generated by the Cooperative Voronoi A* algorithm. The results of planning of this set-up by the Cooperative Voronoi A*algorithm is shown. Preference is given to the central lanes over the outer lanes. The 7 robots follow their generated paths without collision amongst each other and obstacles in the scene.

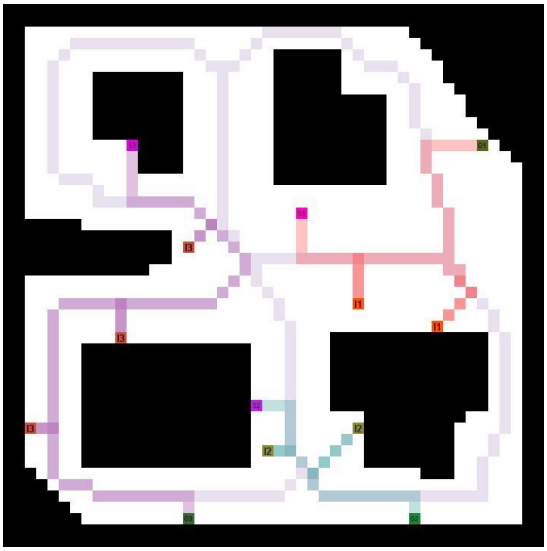

Figure 18: The start, waypoint and goal positions for the different robots that are not on the Voronoi map are connected to their closest points on this map via the Cooperative A* algorithm.

Figure 19: Multiple Voronoi lanes are used to allow robots to pass each other. In this figure 3 lanes are shown. The middle one will be preferred when planning the robots paths.

Summary

Two variations of a cooperative planning algorithm have been implemented: Cooperative A* and Cooperative Voronoi A*. These are a planning based approach to solve the multi-robot coordination problem. A plan is calculated for each robot individually by means of collecting information from the environment and from the positions from the other robots. Visual feedback of the robots moving on their resulting paths is provided through the path planning simulator. The simulator allows for user interaction through map selection, task specification and various visual and configuration settings.

Depending on the planning application many different decisions can be made with regard to environment representation and search algorithm. Some will be more appropriate then others for varying problem areas and circumstances. Within the choice of a specific representation and search algorithm, there is still room to explore the path planning method chosen. Different characteristics of the planning method can further be tailored to the application. Some properties that effect robot behaviour were handled: speed/optimality trade-off, traversibility of the terrain, reducing search space to Voronoi diagram, using distance transformation for safer paths …

Future Work

For partially unknown environments, there isn't a complete and accurate model of the environment. This can be the case for exploratory robots, or when the map is subject to dynamic changes. The D* algorithm (Stentz's algorithm [2]) is capable of planning paths in unknown, partially known, and changing environments in an efficient, optimal, and complete manner. Instead of using A*, D* would expand the cooperative planning algorithms reported here to a larger problem space.

As computations for large maps become increasingly time-consuming, alternate representations of the environment can be considered, for example adaptive cell decomposition. As opposed to regular cell decomposition, adaptive cell decomposition will result in maps with varying cell sizes. This approach reduces the number of cells used in open areas, which leads to less space for memory storage and less computation time. This would especially be advantageous in terrains with large areas that have the same traversibility. However, when new map data becomes available, updating such a map is less straightforward, as the entire map structure may need altering.

Acknowledgements

This work was partially supported by the research project VIEW-FINDER, FP6-IST- IST-045541 from the European Committee.

References

[1] David Silver, "*Cooperative Pathfinding*"

[2] Anthony Stentz, "*Optimal and Efficient Path Planning for Partially-Known Environments*", In Proceedings IEEE International Conference on Robotics and Automation, May 1994.