Application of Traversability Maps in the Virtual Rescue competition

Maarten van der Velden Wouter Josemans Bram Huijten Arnoud Visser

***Abstract*— This paper describes an exploration behavior which has been designed to explore large indoor areas. While exploring, traversability information is saved onto a layer in the map. This information is used in an A* algorithm has been implemented and is used for path planning. This approach allows the robot to learn from its experience and to find in the long run the "ideal line" to driving through the indoor environment.**

I. INTRODUCTION

Something that robots have to deal with continuously while they move around is the terrain they drive on or walk over. Some types of terrain are very accommodating to a robot, while others impede the robots in its tasks or can even prove to be hazardous to move on. Usually, most sensors are dedicated to detecting obstacles. This means that they usually "watch" for obstacles at some height in a line parallel to the ground. Only in some cases are sensors used to look at the terrain itself. An example of such use is found in the autonomous vehicle Stanley [1], which uses laser range finders to analyze the terrain ahead.

It would be ideal; however, if robots that have not been equipped specifically for the purpose of measuring terrain quality could provide some kind of indication of how well the terrain can be traversed. These robots could contribute to building a map of the terrain, which could then be used to plan a path from one point on the terrain to the next. This path can then be safely followed by the robot, and hopefully it will allow the robot to reach its goal faster than it would by using traditional path planning.

What we would like to research is: does using a traversability map improve the efficiency in which a robot can complete its tasks? We hypothesize that this is the case. The robot should be able to avoid terrain that would slow it down significantly and the robot should be able to find some kind of "ideal line" when driving through e.g. a corridor. This ideal line should then minimize the loss in driving speed due to directional changes.

This study is performed in the USARSim simulator [2], the environment used in the RoboCup Rescue Simulation League virtual robot competition[1]. This simulation environment is a research framework used to simulate robots and sensors on a physical realistic level. In Fig. 1 a screenshot of a world in this simulation environment is given. There are several types of robots the user can pick for his simulation. In this study a P2AT robot and the DM-Mobility_250 map were used.

Intelligent Systems Laboratory Amsterdam, Universiteit van Amsterdam, Science Park 105, 1098 XH Amsterdam, The Netherlands

[1]USARSim can be downloaded from http://sourceforge.net/projects/usarsim

Fig. 1: Screenshot of one of USARSim's competition worlds.

Advantages of the USARSim environment are:

- Physics simulation: the framework makes sure that robots fall down when they drive over a cliff.
- Realistic simulation: the robots and their sensor in the simulation are just as imperfect as real life robots; their sensors contain noise and their movements are often an approximation of the requested movements due to friction between wheels and ground.
- Real-time, realistic visual feedback: you can follow the robot visually as it drives around the map, or you can float around the map at your leisure to plan a challenging route for the robot.
- Reproducible results: the framework allows the development of robot control algorithms under controlled circumstances.

II. RELATED WORK

Howard *et. al* [3] already used traversability maps for a P2AT, yet the terrain was the JPL Mars Yard with many small obstacles. The traversability was not based on experience, but on extracted from characteristics extracted from visual imagery data. The traversability map was converted into gradient fields, which was used in a path-planning algorithm to select both the optimal path and speed.

Ye and Borenstein [4] estimated the traversability by converting an elevation map, obtained by a laser scanner. Each terrain patch was analyzed both on roughness and slope. The traversability information was directly used inside steering and speed control. No path-planning was performed.

Seraji [5] estimates traversability information both from experience and long range sensors. This information is stored in two different maps (local and global). The robot is controlled by path-planning, but integration of multiple turn-

and drive-behaviors. The applicability of the approach is tested in a 2D-simulation environment.

Traversability maps are not only used in space applications. A recent application was by Crane *et. al* [6], who used traversability grids to navigate their vehicle during the DARPA challenge. They were able to use a detailed grid while maintaining real-time performance.

III. METHOD

This section is divided into three parts.

A. Exploring traversability

The problem with determining traversability in an unknown environment is that the robot does not know on what underground it is moving. In the context of this study we do not consider options to determine the quality and type of the terrain ahead of the robot. So the robot can only determine the traversability on terrain where it has actually driven.

Because many environments made for humans consist of rooms, and robots will often be used in environments created for humans, we have developed a behavior that is optimized for large, room-like structures: the first thing the robot does is drive straight ahead until it encounters an obstacle, it then stops and turns until the way ahead is clear. It then drives again until it encounters another obstacle. If no strange things are encountered, this will result in a completed quadrilateral. The robot will know that a quadrilateral has been completed when it encounters a place where it has already stored traversability on the map.

The robot also stores the information on the points where it has turned so when it knows a quadrilateral has been completed, it knows the corners of the quadrilateral. Let us name these corners with digits: 1, 2, 3 and 4. 1 is the point where the robot currently is and the rest of the points are numbered counter-clockwise. The robot can easily calculate the points on the middle of each of the sides of the quadrilateral. Let us name these points with letters A, B, C and D. Point A lies between 1 and 2, point B lies between 2 and 3, point C lies between 3 and 4 and point D lies between 4 and 1. The robot will now start to explore the space inside the quadrilateral by following a path between these points in the following order:

1(start)-3-A-4-B-1-C-2-D-3(end).

This results in the pattern indicated in Fig. 2:

Because we will interpolate the measured traversability data on the map, this will be sufficient for all but the largest spaces. For small quadrilaterals a few traverses are sufficient. We therefore calculate the number of traverses to be made by dividing the distance between 1 and 3 by a constant factor. In our test environment this factor was 2500, but this can be tweaked to prefer more thorough exploration. In our test environment the medium sized square is traversed 4 times before considering the terrain sufficiently explored.

Another option for exploring the inside of the square was also considered: starting in the center and then creating a search spiral, as often used in outdoor search and rescue missions.

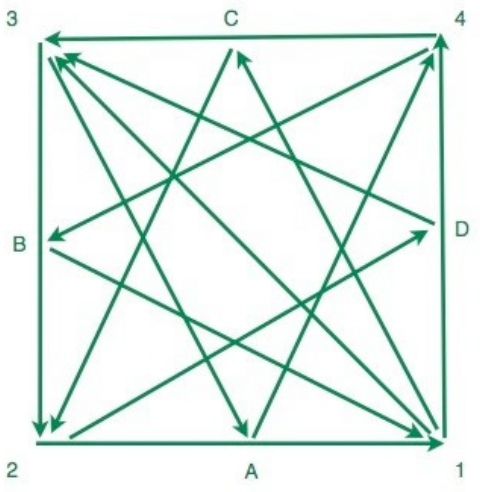

Fig. 2: Search pattern for an ideal square environment.

This would result for our ground robot a path with many sharp turn and because the turning speed of the robot was very low, the time penalty compared to the current behavior would be severe, and this was therefore not implemented.

In the behavior we developed, the robot switches to frontier exploration when the last traversal has been made. Frontier exploration for these robots is not implemented in the classical way [7], but with the *beyond frontier* algorithm [8]. In this algorithm a selection is between a small number of frontiers, based on the distance to those frontiers **and** the information gain available beyond those frontiers. This will result in the robot planning a path exiting the room and entering the hallway. If it enters another large room, it should switch back and restart exploring using the method described above.

B. Measuring and mapping

The goal of the exploration behavior is of course to gain information considering the traversability of the terrain explored. In this study the goal was to collect this data without using visual imaging data. We made an estimate of the traversability of a location by comparing the forward speed the robot is commanded to make (by the motion algorithm currently running), with an estimation of the actual speed the robot makes using location sensors (which can be any one of odometry, GPS, etc.).

The formula used to define the relation between both speeds and the traversability is:

$$T(x) = \frac{v_{REAL}(x)}{v_{GIVEN}(x)} \tag{1}$$

where $T(x)$ is the traversability at location x, v_{REAL} is the estimated real speed and v_{GIVEN} is the given speed of the robot at location x.

The estimation of the traversability could be enhanced by using more parameters, such as data from the robot's inertia sensors, but for the course of this study this enhancement was out of scope.

Using the formula above, the robot will slowly aggregating traversability data. This information is stored as a function of the location on the map. An important design decision is the granularity of this storage. With a high level of detail represents only the trajectory driven by the robot.

It would be better to extend this information and to make an estimation of the traversability of nearby unvisited places. It is important to spread this information not too far. A logical boundary is to stop the spread at obstacles visible to the robot. These obstacles can be accurately detected by a laser-range scanner.

Making estimations of unvisited terrain in between the robot and the known obstacles involves a more sensible representation of traversability with two layers. Before, the traversability value was indicated as a number between 0 and 255 in the red channel of the map. Additionally, storage is needed to indicate the certainty of the estimate. The green channel was used to indicate certainty of the traversability value.

Following this new approach, the traversability values generated at the locations where the robot really has been get a certainty value of 255 (maximum), while the certainty of the 0 traversability values of obstacles is set at 40%. In this way measuring errors in the laser-range scanner or in the simulation environment won't be overestimated, only when an obstacle is 'sighted' a couple of times the certainty will be 100%.

Now the estimation of traversability of unvisited terrain. On default, all locations we don't have any data on will have a certainty of 0 and will therefore not be calculated. All area that becomes within range of the laser-range scanner (except of the obstacles) and the locations already known will get a certainty and a traversability of 50% both. These values are not changed with more scans, because you don't know more when you see 'no obstacle' (because that's exactly what the laser-range scanner 'sees') twice instead of once, when it's still far off.

The information, together with the obstacles is saved to a layer in the map (note that the actual trajectory of the robot was is saved in another layer).

So there are two layers, but the information stored on these layers is rather sparse. To improve the usability of this information an interpolation is made by means of a convolution with a Gauss of the visited places over the map with traversability of the area. This results in the following image:

In this way the area surrounding places already visited changes color towards the certainty and the mobility of the location visited. Doing this, a gradient is being made showing rather detailed what the mobility at a certain location on the map is and with what certainty it can be stated.

To show the traversability at known places we used the existing infrastructure in USARSim to make a mobility layer in the existing manifold map. This makes it rather easy to make the collected data available for other robots at the same time in the same environment or to save the information to be used by robots at a later time. In this way the traversability

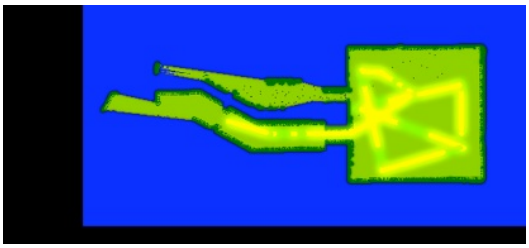

Fig. 3: Interpolation outside the driven trajectory by means of a convolution with a Gauss.

of a map is slowly learned. How robots can benefit from this information is worked out in the next section.

C. Path Planning

Our path planning approach is based on searching through a state space represented as a graph. Graph search is a well known technique, although it is here applied in a realistic setting. In this setting there are several issues that are not directly related to graph search, but need to be addressed for a robust performance.

In order to use the data in the traversability map, we augment each node in our graph with the corresponding traversability index. We use this traversability index to determine the cost of passing through the node. A low index means a high cost and a high index means a low cost.

First, there is the issue of having to plan a path to a target in terrain that we have not explored yet. In this case, a portion of the path will always have to be planned with unknown traversability (however, obstacle detection can be used). This means that we will not find the most efficient path in this case. We can, however, add the information gained from driving over this unknown piece of terrain to our traversability map, so that the next time we need to find a path to the target we will have more knowledge about the terrain.

Another issue would be how to weigh the cost of taking risks. For instance, look at the situation in Fig. 4:

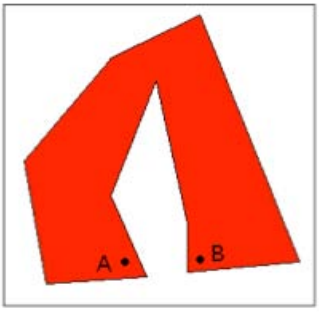

Fig. 4: Situation where taking a safe route will result in a detour.

Here, the red area is terrain for which the traversability is known (the image is not a traversability map, but a binary form of the confidence level). The robot has to plan a path from point A to point B.

Obviously, the shortest path between A and B would be driving in a straight line. The terrain between A and B could be perfectly safe to drive over. Then again, it might also be terrain that is completely untraversable. The issue here is whether to be safe and take the longer route over known terrain or take the risky shorter path. This issue is similar to the "exploration versus exploitation" issue that is often addressed in the field of reinforcement learning.

In order to incorporate the choice between these two approaches into our framework, we decided to assign all unknown terrain some traversability index κ. In order to choose a reasonable κ, we let its value range between two extremes. If the robot is performing a task that requires it to optimize on speed, then κ should be close to 1 (a κ close to one means that all unknown terrain is assigned a high degree of traversability). On the other hand, if safety is the top priority of the robot, then it would make sense to choose a low κ (unknown terrain will have high cost). In our implementation, we have given the robot three behavioral modes; a conservative, a progressive and a balanced mode. The conservative mode maximizes the exploitation of current knowledge and prioritizes robot safety. The progressive mode maximizes robot speed and the balanced mode balances speed and safety.

Another important aspect is that κ depends on is the percentage of the terrain that we have explored. If we have explored only a small fraction of the total terrain, then we know relatively little about the traversability so we do not want to rely on this knowledge very much. In this case, we choose a high κ. On the other hand, if we have explored a lot of the terrain, then we can say with a great degree of confidence that our knowledge about the terrain is reliable, so we should award unknown terrain a lower κ. The fraction of the total terrain we have explored we will call e, with $0 =< e =< 1$. The exact definition of κ, depending on the mode, is as follows:

$$\kappa = \begin{cases} 1 - \frac{e}{\epsilon}, & \text{if } e < \epsilon; \\ 0.05, & \text{if } e \geq \epsilon. \end{cases} \quad \text{Conservative mode} \quad (2)$$

$$\kappa = 1 - \frac{e}{2} \quad \text{Progressive mode} \quad (3)$$

$$\kappa = 0.3 \quad \text{Balanced mode} \quad (4)$$

ϵ in this case is a small constant; the value of this constant will have to be determined by testing. We have implemented the A* algorithm to perform graph search. A* assigns a cost to every node in the graph. This cost $f(x)$ is calculated as follows:

$$f(x) = g(x) + h(x) \quad (5)$$

Where $g(x)$ is the total cost from the start to the current node and $h(x)$ is the estimated cost of going from the current node to the goal. This algorithm is implemented by sorting an priority queue[2] with the distance measure $f()$. A description of the A* algorithm is shown in below:

Data: the traversability map m, the start point s, the target point t
Result: the optimal path p from location s to the location t
closed = EmptyList();
open = EmptyPriorityQueue(s);
while *Not IsEmpty(open)* **do**
  c = HighestPriority(open);
  if $h(c,t) < \epsilon$ **then**
    Return $p(c)$;
  end
  if *Not IsMember(closed,c)* **then**
    closed.Add(c);
    for *each neighbor(c,n)* **do**
      $d_n = g(s,c,m) + h(n,t)$;
      $p(n) = p(c) + n$;
      QueueSortAdd(open,n,d_n);
    end
  end
end
Return EmptyList();

Algorithm 1: The A* algorithm for the path-planning with the real travel cost $g()$ calculated on the traversability map, and the heuristic travel cost $h()$ calculated with the Euclidian distance.

A general issue in A* graph search is finding a good heuristic $h()$. In path planning through some space this heuristic is usually the Euclidean distance between a node and the goal. In our implementation, we use this heuristic as well. Our real cost $g()$, however, is not simply the traversed distance from the start to the current node but the following:

$$g(x) = \frac{distance(start, x)}{traversability(x)} \quad (6)$$

As the traversability index is a number between 0 and 1, the real cost will always be equal to or greater than simply the distance. This means that our heuristic never overestimates the cost, and is therefore admissible (A* only finds the optimal path if the heuristic is admissible.)

IV. RESULTS

In this section the individual parts are demonstrated and the integrated exploration behavior. The system measures and stores traversability data and this information is used in the path planning algorithm. Unfortunately, no explicit comparison of the navigation with and without using traversability data is performed. We hope others will continue our efforts to determine the gains to be had when the traversability information is used in the way we think it should be.

Other challenges yet to be answered could be how well the traversability is captured and whether more data (from other sensors) can be used to measure traversability? There are also a number of constants used in our code for which the optimal values have yet to be determined. For example

[2]The implementation provided by Rasto Novotny is used. This implementation is published for download in December 2005 on http://www.developerfusion.com/code/5052/priority-queue-net/.

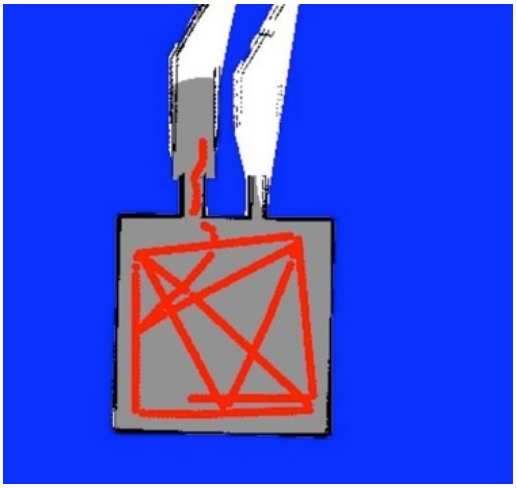

Fig. 5: Resulting search pattern for a square environment.

what is the optimal sigma to be used in the blurring to create the traversability maps? What are the best values to be used? Can these be predicted?

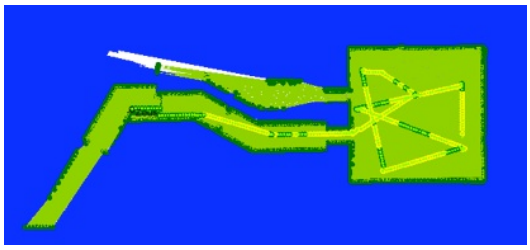

Fig. 6: Known traversability saved along the path of the robot.

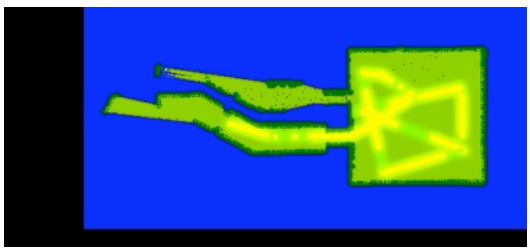

Fig. 7: Traversability map after interpolation of known traversability.

But perhaps most interesting from a 'learning' perspective is to extend the implementation by using more robots simultaneously to gain information faster, and to make better use of the information that is already available.

A quite important point we didn't manage to get to the bottom of, is how well the USARSim environment simulates the physics needed to be able to correctly determine traversability, since even on slopes, the robot would return the same traversability values as on a level surface.

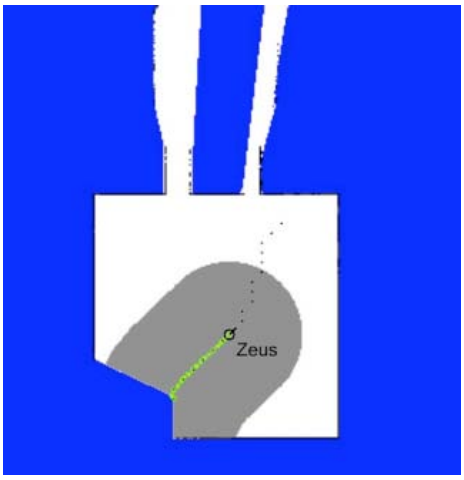

Fig. 8: A path planned using our A* implementation.

V. CONCLUSION

An advantage of this experience based approach is that it does not require very complex terrain models. In fact, the robot does not need to know anything about the terrain at all other than how fast it is moving over it. This means that this approach should work for any kind of terrain, be it in water, air, the desert or the red sands of Mars.

Acknowledgements

We want to thank the other members of the Amsterdam Oxford Joint Rescue Forces[3] for providing us with an extensive framework to experiment with.

REFERENCES

[1] S. Thrun *et al.*, "Winning the DARPA Grand Challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, September 2006.

[2] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "Usarsim: a robot simulator for research and education," in *Proceedings of the IEEE Conference on Robotics and Automation (ICRA'07)*, 2007, pp. 1400–1405.

[3] A. Howard, B. Werger, and H. Seraji, "Integrating Terrain Maps into a Reactive Navigation Strategy," in *Proceedings IEEE International Conference on Robotics and Automation, (ICRA '03)*, vol. 2, September 2003, pp. 2012 – 2017.

[4] C. Ye and J. Borenstein, "T-transformation: Traversability analysis for navigation on rugged terrain," in *Proceedings of the SPIE Defense and Security Symposium, Unmanned Ground Vehicle Technology VI (OR54)*, April 2004.

[5] H. Seraji, "New Traversability Indices and Traversability Grid for Integrated Sensor/Map-Based Navigation," *Journal of Robotic Systems*, vol. 20, no. 3, pp. 121–134, February 2003.

[6] C. D. Crane III *et al.*, "Team CIMAR's NaviGator: An unmanned ground vehicle for the 2005 DARPA grand challenge," *Journal of Field Robotics*, vol. 23, no. 8, Augustus 2006.

[7] B. Yamauchi, "A Frontier Based Approach for Autonomous Exploration," in *Proc. of IEEE International Symposium on Computational Intelligence in Robotics and Automation*, July 1997.

[8] A. Visser, Xingrui-Ji, M. van Ittersum, L. A. González Jaime, and L. A. Stancu, "Beyond frontier exploration," in *RoboCup 2007: Robot Soccer World Cup XI*, ser. Lecture Notes in Artificial Intelligence, vol. 5001. Springer-Verlag, July 2008, pp. 113–123.

[3] http://www.jointrescueforces.eu