Mapping an outdoor environmnent for path planning.

Maite López-Sánchez*, Gaurav Sukhatme and George A. Bekey.
Robotics Research Laboratory
University of Southern California (USC), USA.
(*) Visitor scholar from the Institut d'Investigació en Intel.ligència Artificial (IIIA-CSIC), Spain.

Abstract

We present an incremental map building approach that is applied by a group of cooperating heterogeneous robots. Robots cooperate by sharing information in order to build their own maps. Environment information comes into the mapping process from two different sources: aerial images from a helicopter and sonar readings from several ground robots. Ground robots use the resulting map to plan paths towards goal positions. These paths avoid detected obstacles and are updated when there is new information about an obstacle obstructing them. Finally, we consider environment uncertainty depending on the reliability of the information. We use uncertainty as an estimation of the real existence of detected obstacles and we apply it in order to plan paths that may need to go trough non-real obstacles.

1. Introduction

This research report is the result of working on a specific problem within a larger project. The problem is one of representing and managing information about an outdoor environment in a way that it is useful for path planning purposes. The generic goal of the longer project is to achieve cooperative tasks with a group of heterogeneous ground based and airborne vehicles. In the work presented herein, the different classes of vehicles accomplish cooperation by sharing information about the environment. We consider an outdoor environment which is unknown a priori.
The global scenario of this project is a group of autonomous vehicles that consist of a helicopter and five ground robots that cover a certain area of an arena or reach a specific position specified as a command from a human. Our work has been focused in the representation of the environment as well as in the generation of paths that can be useful for the ground robots to avoid obstacles while reaching the goal position. Considering all the obstacle information that is available at each time, we obtain the shortest paths. However, we do not always use all the information details, therefore we can not guarantee optimality (in terms of distance) although we obtain optimal paths for each level of detail. Nevertheless the utility of a path strongly depends on the reliability of the information. In our approach we have two different sources of information: first, the helicopter has a camera facing the ground that provides a birds-eye view of the arena, and second, each ground robot has seven sonar sensors that detect ground obstacles. Robots communicate the obstacle information they gather in order to complete the map of the environment. Since every robot receives information about the obstacles detected by the rest of the robots, all of them have the same individual maps (or similar, in case of transmission problems). The distribution of information among the robots allows other robots to use information gathered by a robot that reached a specific region before them. Nevertheless, this does not prevent any robot to plan its own path under bad communication circumstances: since a robot keeps its own version of the map it can still plan a path albeit with less information.

In the following section we describe how information from the different sources is preprocessed and used as input data for mapping. The third section gives the details of how our incremental mapping approach is based on the grouping of environment obstacles. The resulting map is used to generate paths, and the fourth section illustrates this planning process. Finally, section number five explains the way in which we include uncertainty into the mapping process.

2. Obstacle information extraction

As we have already said, two different kinds of robots gather and share information from the environment. On the one hand the helicopter captures aerial images and, on the other hand, ground robots detect obstacles by means of their sonar readings.
During the helicopter's flight, image processing will be performed by its exclusively dedicated image system. Figure 1 shows an image of a typical outdoor environment, a parking lot. This image is of the same kind of the ones captured by the helicopter camera. Since the helicopter chassis development is still in progress, this image was taken from a building (the perspective is the main difference between the neighboring images that the helicopter will obtain and this image) and processed on a sun station. Image processing has been done using Matlab 5.2 and gives as output a set of polygons that contain some of the obstacles in the real world. The reason for using polygons as descriptions of environmental features is that they constitute a simple and compact way of representing information, so that their communication still allows the robots to perform other tasks simultaneously. The following steps describe how to extract polygons from a gray-scale image (Figure 2 shows the results of applying these steps to the image in Figure 1):

1. Obtain the most common gray that appears in the image and consider it as background color.
2. Define lower and upper thresholds so that they specify the range of grays around the background color that are still considered as background.

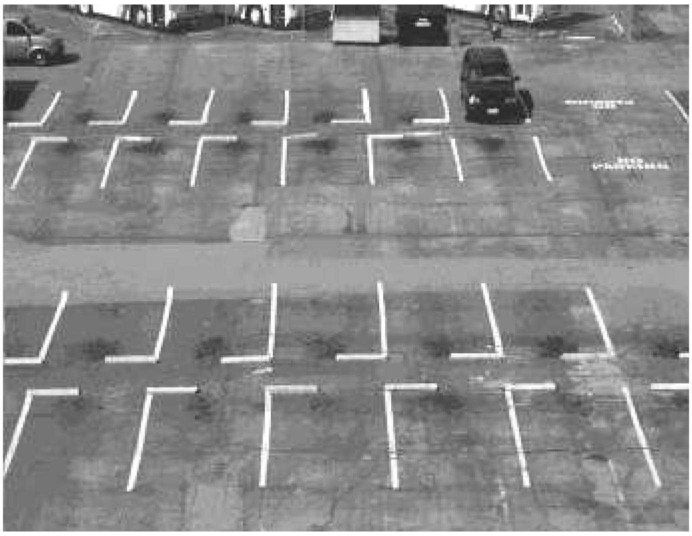


Figure 1: Outdoors environment image

3. Transform the gray image into a binary bitmap setting the background pixels to 0 and the remaining ones to 1.
4. Apply several image processing Matlab functions (erode, dilate, clean and majority) to increase the quality of the image by removing spurious pixels.
5. Identify the areas that result from the grouping of neighbor white pixels.
6. For each of the resulting areas, compute its Convex Hull, which is the smallest convex polygon that includes an area.

7. Compute the area of each polygon and eliminate those having an area smaller than a certain threshold.
8. Polygons are specified as a list of vertices. Some of these vertices can be discarded without causing a significant change in the area of the polygon. (Figure 1 had 60 vertices per polygon on average and was reduced to an average of 20 vertices per polygon).

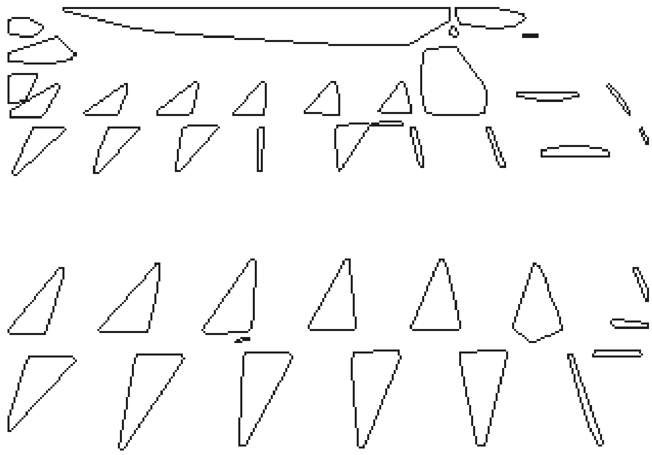


Figure 2: Image processing resulting in polygons

Essentially, this sequence of preprocessing steps separates the foreground from the background and attempts to represent all foreground entities as polygons. Since the helicopter has no stereo vision, the image processing system cannot distinguish real obstacles (in the sense of ground protuberances) from shadows or paintings, therefore there is no guarantee that a polygon will indeed define an obstacle in the environment. See, for example, the painted vertical parking lines which are considered to belong to the horizontal white obstacles in Figure 1. Although, obtained polygons do not thus necessarily correspond to real obstacles, they may represent part or groups of obstacles, they can still be considered as being helpful in identifying areas of potential danger for the ground robots.

The information obtained from the image processing system is incomplete, so the ground robots will need to apply reactive techniques to deal with those obstacles that where not identified in the image but are detected by sonar. This obstacle information is also added to the environment representation and shared among robots. In order to keep a homogeneous representation of the obstacle information that comes from different sources, a robot that is avoiding an obstacle can consider consecutive sonar readings to generate a polygon that approximates the edge of the obstacle. Figure 3 illustrates an example of how a robot follows part of the edge of wall and the resulting polygonal approximation. Basically the polygon comes from the grouping of consecutive readings that are approximated by a line, so resulting polygons are in fact rectangles containing those lines, with different length and orientation, but with a fixed width that has been defined by default. In fact sonar readings are filtered in such a way that distant readings and groupings that yield segment lines shorter than a threshold are not considered.

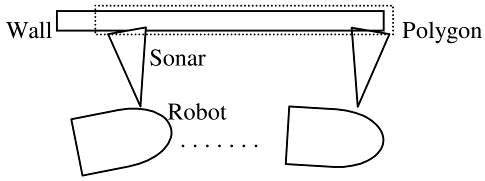


Figure 3: Ground robot obstacle detection

Although this is beyond the scope of this work, linear grouping of sonar readings is used when the same wall or obstacle edge is detected more than once. Two thresholds (angle and distance) define when two linear groups of readings are considered to come from the same obstacle. When this is the case, comparing the orientation of the first detection with the most recent one shows how the orientation error has increased through time. Therefore, the latest segment detection can be rotated to match the same orientation as the first detected segment, and this can be applied to subsequent detections. Once two segments are parallel, the distance between them is used to correct the position of the robot (it is translated in the perpendicular direction of the reading).

3. Mapping

The previous section defined how obstacle information has been retrieved and specified by polygons, which constitute the input data of the mapping task. This section describes how these polygons are used to incrementally create a map of the environment. Since each robot has a map, every robot updates its own map whether it receives a polygon detected by another robot or it detects the obstacle by itself (and then broadcasts it as well). Thus, all robots apply the method described here simultaneously.

3.1 Related work: Map representations

When choosing a representation of outdoor environmental features, there are several characteristics we must take into account. On the one hand, outdoor environments are especially difficult to map because the shape and distribution of obstacles are less predictable than human-made obstacles in indoor environments. This makes it difficult to identify and describe obstacles, particularly, when the information comes from a single camera and local sensors embedded in robots with dead-reckoning errors. On the other hand, it is important to deal with the uncertainty associated with each map element when choosing the right map representation. In indoor environments, a grid representation is the most commonly used [3,4,5,6]. Nevertheless, it is not flexible enough in terms of obstacle positions or contradictory sensor information when used for outdoors environments. The reason is that, for example, if an obstacle that corresponds to more than one cell is represented in the grid and afterwards new information suggests that the obstacle representation should be associated to other cells in the neighborhood, there is no a direct way of moving it. Furthermore, if we want to maintain resolution, then the representation size increases with the size of the environment. This is not the case for a symbolic representation of the environment such as a graph with obstacle areas as nodes and their relations as edges [2,8,10]. In such a graph representation, location is just a characteristic -that in some cases can even be reduced to be implicit in node relations- and size depends on the number of obstacle areas, which keep the problem tractable when outdoors.
It is also crucial to have a task oriented representation, that is, to represent information in such a way that it is easy for the robot to use it for performing its task. In our case, the ground robots will use the map for planning paths that go from their initial positions to the goal. Graph representations have well known optimal path planning algorithms in the literature. This reason, in addition to the previous ones listed above, makes us choose a graph representation for outdoors mapping. In particular, our map consists of a Visibility graph [9]. This kind of graph representation assumes that obstacles in the environment can be approximated by polygons so that every vertex of each polygon is used to define a node in the graph. Edges are then defined between 'visible' nodes, that is, those node pairs without any obstacle in between. The edges are labeled with the distance between the nodes. An edge means that it is possible for a robot to go from one node –or obstacle vertex- to the other covering the distance in a straight line without finding any obstacle in its way.

3.2 Our approach: grouping obstacle information into the map representation.

Considering the polygonal information that the robots gather from the environment, the process of building a Visibility graph is direct although the visibility check for all node pairs –i.e. polygon vertex pairs- may be unnecessarily expensive in terms of computational time. The purpose of our mapping task is not to give a detailed description of an environment, but a rough approximation of those environmental features that can be a potential danger for the ground robots when approaching a target position. What we propose is to build an efficient higher level abstraction on top of the polygons such that a robot is able to plan its path towards a goal position at this level. Nevertheless we keep the relation between both levels of representation so that if occasionally the robot needs a more accurate path, it can go down into the polygon level and still use the abstract level to simplify the amount of treated data. In general this should not be the case because outdoors environments usually have a relatively low obstacle density. This makes unnecessary for the robots to know the exact shape of an obstacle in order to avoid it.

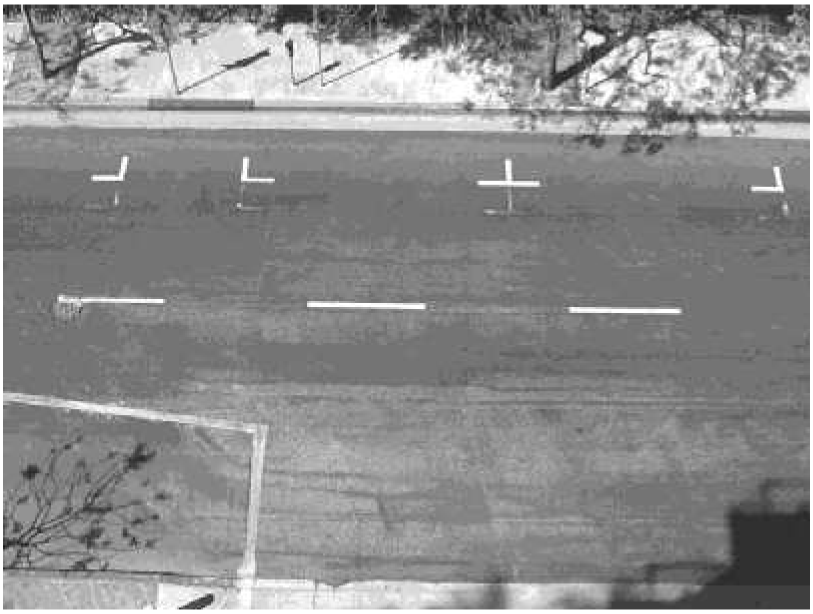


Figure 4: Outdoors environment image

We characterize the higher level of abstraction as a set of *Obstacle Areas*. An obstacle area being a rectangle that includes a group of overlapping polygons. Polygons can intersect due to different reasons: on the one hand the vision system can provide polygons that overlap, and on the other hand, an obstacle –or part of it- can be detected several times, by different means or from distinct perspectives. Thus, map information is distributed over two levels: the first one defines how the obstacles in the environment are distributed in different areas, and the second level goes into more detail and specifies the polygons that belong to each area. Figure 4 presents another example of an outdoors image and Figure 5 shows how their corresponding image processing polygons have been grouped into rectangular obstacle areas.
Figure 5 gives a good description of the kind of obstacle areas that we can find. Isolated obstacles in the environment yield obstacle areas with a single polygon, in these cases the gain comes from the fact that we reduce an average number of twenty vertices per polygon to a constant number of four. This might not seem a big improvement in the performance, but it has the advantage that it does not dismiss important information. Since it is an isolated obstacle the robot will still have room to surround the obstacle area rectangle. Bigger obstacle areas are

usually composed by several overlapping obstacles. In these cases they tend to cover bigger free space areas -which increase the risk of covering useful areas for path planning. However the reduction of the number of vertices grows in proportion to the number of included polygons. Nevertheless, the vertices of those polygons that are completely contained by others are discarded without taking any additional risk.

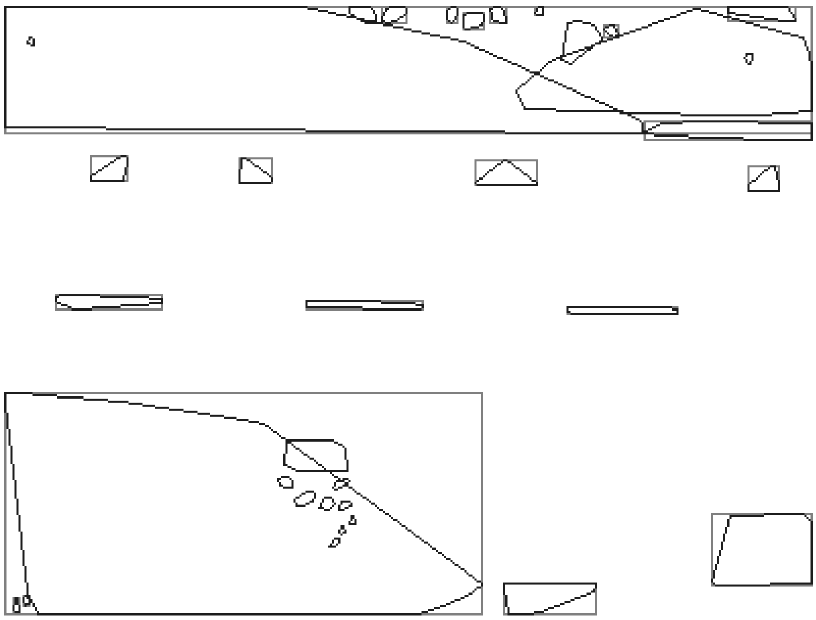


Figure 5: Grouping obstacles: obstacle areas are defined as rectangles containing intersecting polygons

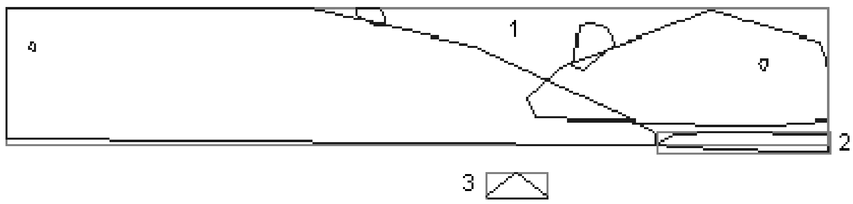
Finally, there is a special kind of obstacle areas that are produced when polygons do not intersect but the rectangles of their corresponding obstacle areas intersect. We call them *intersecting obstacle areas*. Usually they require the system to go down to the polygon level to treat the relations among them. Intersecting obstacle areas are the ones that provide less efficiency gain because going to the polygon level means getting closer to the original computational costs, but in the following sections we will show that we still have some gain in such cases.

3.3 Map structure

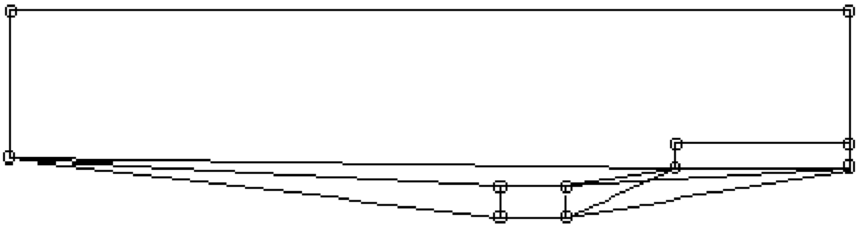
Our map representation consists of a set of obstacle areas and a visibility graph:

- *Obstacle areas* present two levels of information: the highest level specifies a rectangle that contains a set of intersecting polygons, and the lowest level includes the list of these intersecting polygons. In case an obstacle area is intersecting with others, the list of references to these areas is also stored in the high level. For the low level, it can also compute an additional polygon representing the union of the polygons in the obstacle area. Union polygons are computed under request when refining the path that has been obtained in the high level representation; we will see how they are useful for visibility computations.
- We create the *visibility graph* by considering as nodes the vertices of the obstacle area rectangles and by establishing edges between 'visible' nodes: those node pairs without any obstacle in between. Edges are labeled with the Euclidean distance among the nodes they relate. In our approach, visibility is computed considering the obstacle area rectangles in stead of the polygons.

Figure 6 illustrates how we represent the map information. Figure 6 a) shows a subset of three obstacle areas from the nineteen obstacle areas in Figure 5. Each area has been drawn as its rectangle and the polygons it groups, (for example, obstacle area number 1 groups six obstacles).



a).

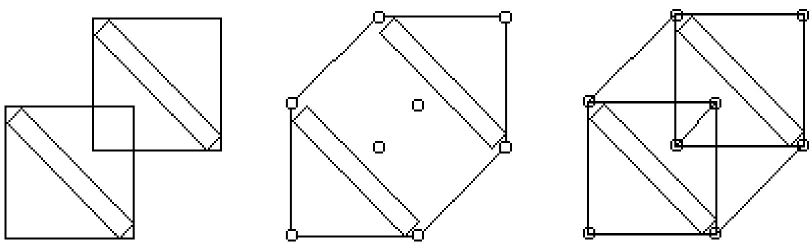


b)

Figure 6: a) Subset of obstacle areas from Figure 5, b) its corresponding visibility graph.

Although it has not been shown in the figure, obstacle areas number 1 and 2 have a cross-reference between them that states their intersection. Figure 6 b) is the corresponding visibility graph. Small circles represent nodes in the graph and come from the vertices of the obstacle area rectangles. However, not all the vertices are included because there are vertices in the intersecting areas that may lie inside an obstacle of one of the areas it intersects with. This is the case of the lower-right vertex of the rectangle in area 1, which is located inside the polygon of the second area.

Keeping the level of abstraction at the obstacle-area-rectangle level speeds up the computation of the visibility. The gain depends on the number of polygons that each obstacle area is grouping and the number of vertices of each polygon. But the gain can be significant if we take into account the fact that all node pairs check for every obstacle area if the area obstructs their visibility.



a) b) c)

Figure 7: a) intersecting obstacle areas, b) checking visibility between nodes considering obstacle area rectangles, c) visibility considering polygons.

Unfortunately, the high level of abstraction may yield a low connection between nodes that belong to intersecting areas. This is not the case of the intersecting areas in Figure 6 but for a case like the one shown in Figure 7, it is necessary to check visibility at the polygon level because the higher level does not connect nodes between the two parallel obstacles.

3.4 Map updating

After determining how data is represented, we describe how robots share information and how they include new information into the current map. The idea is that each time a robot defines a new polygon it broadcasts it to the rest of the team. Thus, all robots are able to update their maps. To add a new polygon into the map means that it must be included into one of the obstacle areas and that the visibility graph must be updated. Graph update is computationally expensive, because it implies including new nodes to the graph and checking if the new polygon obstructs the visibility of those pairs of nodes that were visible before the polygon arrival. However, this process should be kept as cheap as possible because there might be some information that would turn out to be useless for some robots –that is, information about obstacles that are located far away from one specific robot trajectory. We accomplish that by considering again the high level of information. The following pseudocode depicts the updating algorithm and in the rest of this subsection we will see how Update uses the high level of information (i.e., obstacle area rectangles) to restrict the computation.

Let *Map* be a list of obstacle areas $OA_{i\ (i=0..n)}$, where each obstacle area contains a list of polygons $LP_i = \{P_1,..P_k\}$ and let G be the visibility graph associated to *Map*. Then we apply the *Update* function to include a new polygon *P'* into *Map*.

```
Update (Map, P')
{   list_of_intersecting_areas = ∅
    last_area_with_polygon = ∅
    polygon_added = no
    Repeat for all OA_i obstacle areas ∈ Map
    {  intersection = Check_Intersection(P', OA_i)
       If (intersection = intersects_a_polygon_of_LP)
       {  If (last_area_with_polygon = ∅)
             Include_polygon_in_area (P', OA_i)
          Else
             Fuse_areas (last_area_with_polygon, OA_i)
          last_area_with_polygon = OA_i
          polygon_added = yes
       }
       Else
         If (intersection = only_intersects_the_rectangle)
           list_of_intersecting_areas = OA_i
    }
    If (polygon_added = no)
    {  OA' = Create_new_Obstacle_Area (P', list_of_intersecting_areas)
       Add_Obstacle_Area (Map, OA')
    }
}
```

Initially, the Update function checks if the new polygon *P'* intersects with any of the obstacle areas OA_i. An increase in efficiency comes from the fact that when *P'* does not intersect with the rectangle that is defined for an area OA_i, there is no need of checking the intersection for the

polygons inside ($LP_i \in OA_i$). When *P’* intersects the rectangle and it overlaps polygons of LP_i, then LP_i must contain *P’*: if OA_i is the first area that intersects *P’* then *P’* is added to LP_i, otherwise it means that *P’* has already been included into another $OA_{j,\ j\neq i}$ and both areas must be fused in order to have all overlapping polygons grouped under the same area $OA_j' = OA_j \cup OA_i$ (with $LP_j' = LP_j \cup LP_i$). Another advantage of the high level information comes when a polygon is included into an area OA_i in such a way that its rectangle does not grow. In this case, the associated graph *G* that corresponds to the previous *Map* is still valid and does not need a review of its visibility (although *G* may need it when OA_i is an intersecting area).
Information about *P’* intersecting the rectangle of OA_i without overlapping polygons of LP_i, causes the reference of OA_i to be stored in the variable list_of_intersecting_areas. In that way, when *P’* will be added to another area $OA_{k,\ k\neq i}$ all areas (i.e., OA_k together with the ones in list_of_intersecting_areas) will become intersecting areas. Finally, if after checking the intersection of *P’* with all obstacle areas OA_i, *P’* has not yet being included, a new obstacle area will be created and included into the *Map*. Notice that, in fact, this algorithm can be used to update empty maps because it simply creates a new obstacle area for the first polygon (and updates G correspondingly). Hence, this is the function that we use to incrementally create our map.

4. Path planning

When using a graph, the shortest path between two positions is given as a list of connected nodes in the graph so that it is possible for the robot to go from the initial point to the goal following the links between the positions that the path nodes represent. In general, the initial position of a robot and its goal do not coincide with the nodes in the graph. Therefore, to compute the path between two positions, it is necessary to first include them as nodes into the visibility graph (and, of course, establish their connections by computing their visibility). Then, the A* algorithm [7] is applied to obtain the path. A* is optimal for this problem because we can use the Euclidean distance as a heuristic and its triangular inequality property makes it conservative.
Visibility graphs give shorter paths than those methods (such as Voronnoi Diagrams) that try to keep the robot as far away from all obstacles as possible. In fact, when the shape of obstacles is well known and can be approximated by polygons, the method yields optimal paths. Unfortunately, we can not guarantee optimality because of the robots' limited obstacle sensing, but we can come as close as the information allows us to be. If paths that are close to obstacles present a danger, then it is always possible to grow the obstacles by half the width of the robot, to provide some margin.
Obstacle expansion is also useful to guarantee paths that can be actually traversed by the robot. This step can be done in the data acquisition stage, however, we have not done it because we deal with polygons that do not correspond to real obstacles and growing them can only yield to poorer paths. Our underlying idea is therefore to have robots that are able to compute an approximated path towards a goal position, and then a robot can roughly follow the path, applying reactive techniques when its internal map does not correspond to the environment that it is actually sensing.
Figure 8 shows an example of the kind of paths that result when applying the planning algorithm to the visibility graph G. Since G has been created considering the higher level of information, the resulting paths go trough the vertices of the obstacle area rectangles.

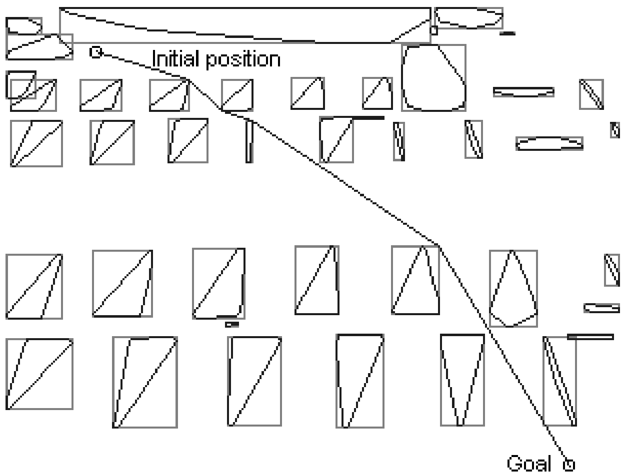


Figure 8: Path that considers high level information in the map from image in Figure 1.

4.1 Path update

Adding information into the map does not necessarily mean changing the current path. In fact, a robot can be following its own path and receiving simultaneously information about other locations in the environment without changing its trajectory. Only information about obstacles that are obstructing the current path triggers the path planning algorithm. Figure 9 illustrates how the path is updated when the information in Figure 2 was not received at once. The left part of Figure 9 shows the path generated by a ground robot that is located at the initial position and has only received twenty polygons from the helicopter image processing system. The path in the right side of Figure 9 has been generated after receiving ten more polygons from the image. The final path that considers all forty-two obstacles extracted from the image in Figure 1 was already shown in Figure 8.

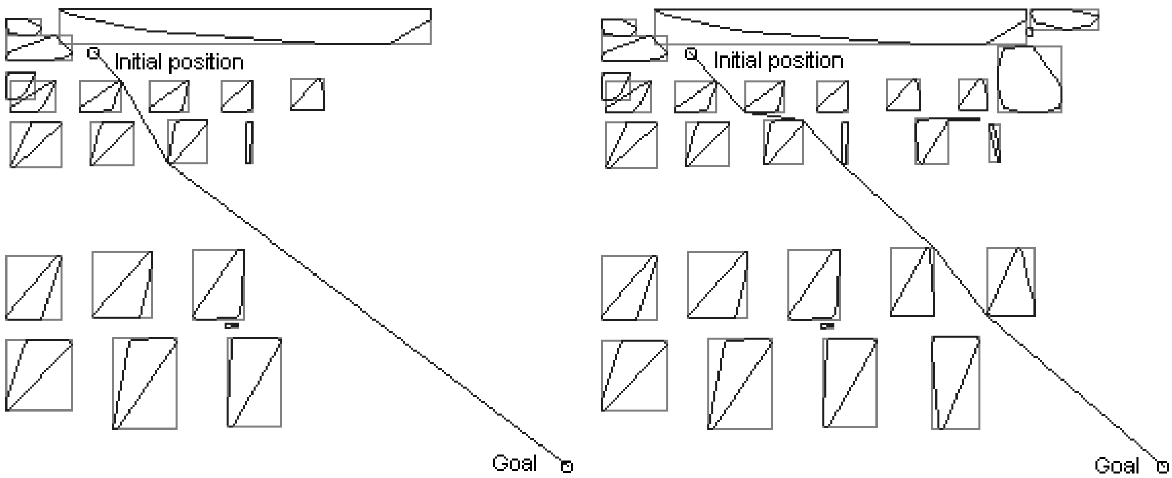


Figure 9: Left: path planning considering twenty obstacles; Right: path after adding ten new polygons

Figure 10: image that might need a path refinement

4.2 Refining the path

In the previous sections we have seen how we can consider the higher level of information and still obtain useful paths. Nevertheless, when the image processing provides obstacle polygons that are a rough approximation of the real obstacles and the resulting obstacle areas cover most of the space in the environment, it may be worth to work at the lower level of information without losing more accuracy. Figure 10 is an example of such situation; the extracted obstacles and the resulting obstacle area rectangles are shown in Figure 11. In this figure the path obtained considering the high level of information has been displayed together with the path that uses the lower level of detail. They have been labeled *path* and *repath* respectively. As expected, the use of more accurate information results in a refined path that is shorter than the higher level path. In fact, this is always the case and the gain depends on how accurate the rectangle areas approximate the obstacles they contain (notice that he gain is 0 for those paths that avoid obstacles going trough vertices that coincide with the rectangle vertices).

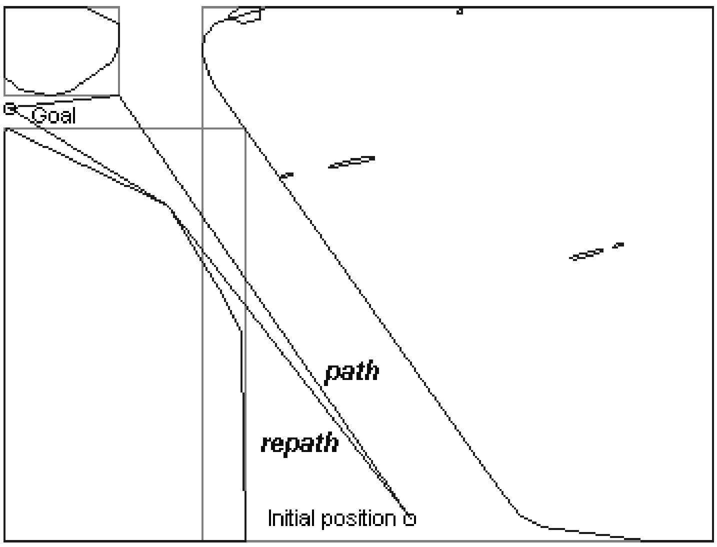


Figure 11: planning at two different levels: *path* considers obstacle areas and *repath* avoids obstacles.

When going from the high level of information down to the lower level, what we are doing is to generate a new visibility graph that has as nodes the vertices of the obstacles instead of using the vertices of the obstacle area rectangles. Figure 12 shows an extreme case that illustrates the difference. In Figure 12 a) two obstacles are included into their respective obstacle area rectangles. The problem is that some of the rectangle vertices lay inside the obstacles, and therefore, they are not included as nodes into the visibility graph. Therefore, the visibility graph that results from considering the high level (see Figure 12 b)) can not generate any path going between the two obstacles. On the contrary, Figure 12 c) shows how this is possible when the obstacle vertices are used.

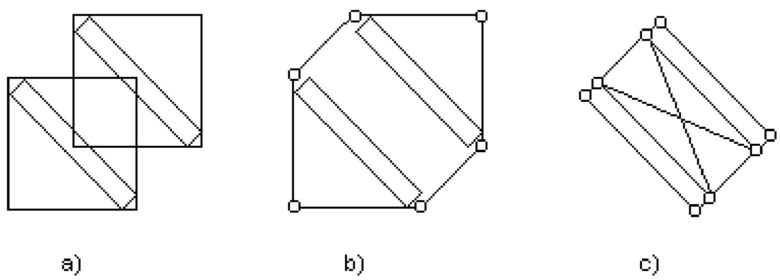


Figure 12: a) two obstacles and their corresponding obstacle area rectangles, b) the resulting high-level visibility graph, c) visibility graph at the low level.

Planning at the polygon level of detail is a computationally expensive process that means to create a new graph including all polygon vertices as nodes and computing all the visibility between them. Fortunately, we can still use the path obtained from the high level planning to choose the obstacles on which planning should be focused.

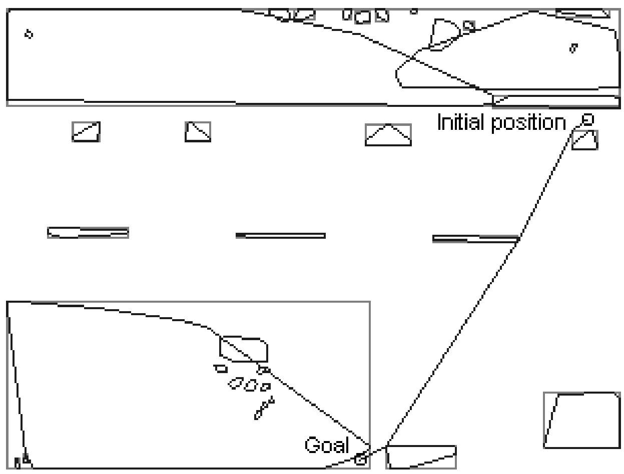


Figure 13: planned path in the map from Figure 5

The basic idea is that information from a less accurate path can help the planning process to discard polygons that are far away from the path. Therefore our approach is to always compute the plan at the higher level, and if required, use it to create a local map in which to apply planning. This local map is built using the obstacle areas that contain the nodes in the path. The difference now is that instead of including as nodes the vertices from the obstacle area rectangles, we include the vertices of the union polygon. The union polygon is an optional part of the obstacle area that is now computed by combining all the polygons inside the area. For

those areas having more than one polygon, this can reduce the number of vertices to include into the graph, especially for polygons contained in others.
As another example, we can use the map from Figure 5 to plan a path between an initial and a goal position. Figure 13 shows the first path that has been computed considering the higher level of information. In that manner, we use this path to choose the four obstacle areas that are used to build the local map that appears in Figure 14. In this case, the refined path is very similar to the high level path due to the accuracy of the rectangles in approximating the obstacles of the involved obstacle areas. We have chosen this example to illustrate the amount of obstacle areas that can be discarded when building the low-level visibility graph (which appears on the right side of Figure 14). On the left side of Figure 14 we can see the union of the polygons of the involved areas and how the refined path goes closer to the obstacles than the one in Figure 13.

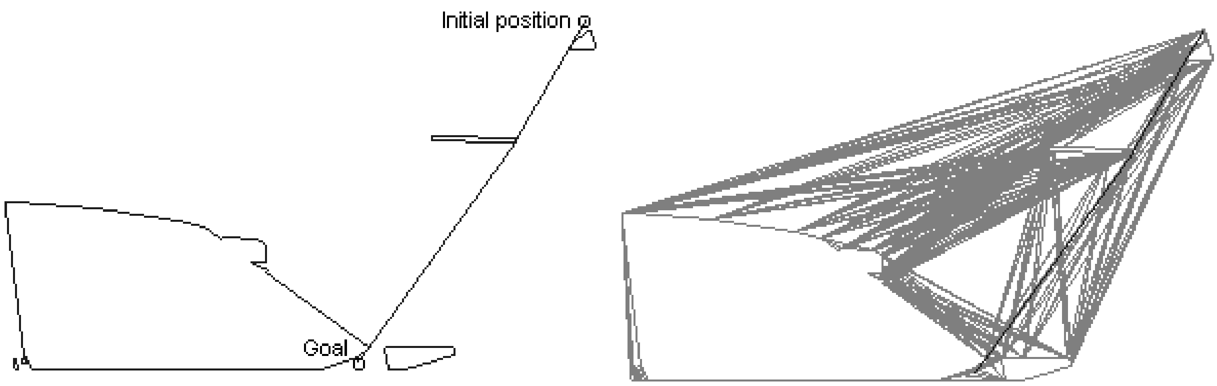


Figure 14: Local map: refined path from the path in Figure 13 (on left) and its visibility graph (on right)

It is not always the case that the refined path that results from the planing on the local map avoids all the obstacles in the global map. In order to guarantee that, it is necessary to go up to the obstacle area level and verify that there are no obstructing global map areas. If there is any obstructing area, it is included in the local map and the local planning process is repeated.
In addition to the refinement of paths, both Figure 11 and Figure 13 illustrate another especial situation of the path planning. In cases where the rectangles of the obstacle areas cover bigger areas than the obstacles they contain, it is possible that the initial or final points of the path lay inside an obstacle-area rectangle without being located inside any obstacle of this area. As we have already said, the corresponding point must be included into the visibility graph establishing the relations with the rest of the nodes in the graph. The difference now is that we compute the visibility by using a combination of both levels of information. On the one side, we consider the obstacles inside the obstacle area that contains the point and, on the other side, we still use the high level of information –i.e., the rectangles- for the rest of the obstacle areas.

5. Considering Uncertainty

When building the map of an environment, there can be different sources of uncertainty, for example, odometry and sensor errors are the main cause of uncertainty in obstacle positioning. However, if we consider that in our approach obstacle information is mainly extracted from an aerial camera, uncertainty about what is and what is not an obstacle becomes the main concern.

Shadows, different materials or color changes are just some of the environment features that can lead the vision system to the identification of wrong objects. One way of facing this uncertainty is to associate with each polygon a certainty degree of its correspondence to real obstacles in the environment. We obtain the certainty degree by taking the product of the area of the polygon and the reliability of the sensor that detected the polygon. In our case we have two kinds of sensors – a camera and sonar sensors- and we assume reliability as being less than 1 for both sensors. In the same manner, we compute the certainty degree of an obstacle area by adding certainty degrees of all polygons inside this area.
The addition of uncertainty about the existence of an obstacle changes the concept of the visibility graph. Considering that a polygon might not be a real obstacle, those node pairs that were not connected because of this polygon visibility obstruction, should now have an edge connecting them.
When planing over a regular visibility graph, the cost of an edge is usually considered to be the Euclidean distance between the two nodes. Considering node uncertainty, more edges must be added in order to represent relations between nodes whose connections might not be obstructed even if the map has some polygons obstructing them. Nevertheless, the cost for these edges must be now weighted distances, with the weights depending on the certainty of the obstacles they go through. The value that we use as an edge cost is the Euclidean distance plus the addition of all the certainty degree of those obstacle areas obstructing the visibility of the edge. Thus, visibility is now checked for the complete graph and edge costs are updated by adding or subtracting certainty degrees.
This new approach generates paths that can go through low certainty areas when there is not a better path to follow. Unfortunately, when the path assumes that an obstacle can be passed through and it turns out not to be the case, the robot that is following the path will need to apply reactivity and will probably end up with a longer trajectory.

Figure 15: Parking image that is partially occluded by a tree.

Figure 15 shows an example of an image that can generate a map with a high density of non-existing obstacles. In this case the image processing system produced eighty obstacles, and most of them do not correspond to real obstacles but to shadows, trees, street lights or painted lines. Although this kind of images may seem useless for the ground robots, we can use uncertainty to plan paths going through these non-real obstacles. Since we have no means of distinguishing real from non-real obstacles, path planning is done over a graph where all non-obstructed edges have a much lower cost than those going through obstacles. In that manner, resulting paths avoid all obstacles that are in fact possible to avoid. The path in Figure 16 illustrates this idea: it goes to the left side of the Figure 15 parking area. This path ends in a position that is free, although a tree makes it to appear occupied. Notice that going through this obstacle is the only way the path can reach its goal. The rest of the obstacles are avoided so that even if some of them do not correspond to real obstacles, those that do correspond are safely avoided.

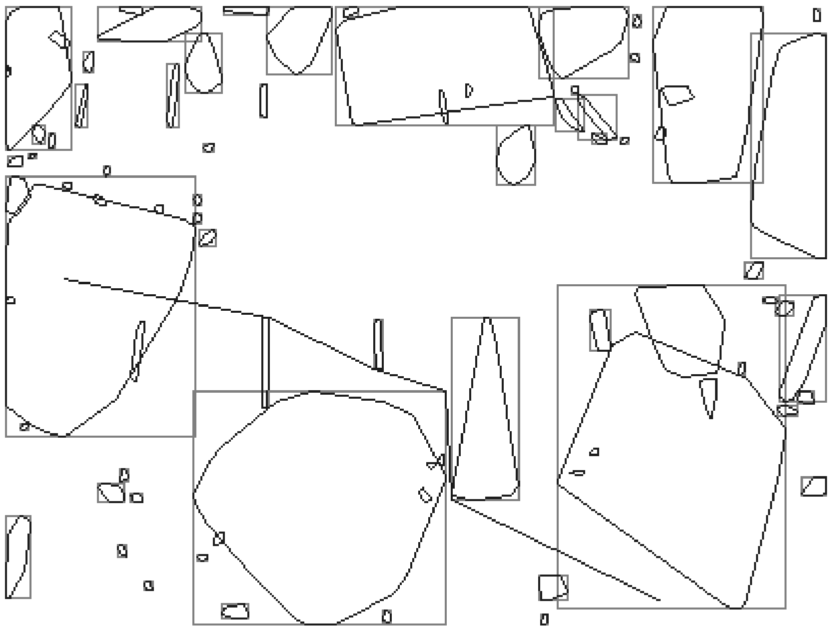


Figure 16: Planning with uncertainty: the path goes trough an obstacle in the map that does not correspond to a real obstacle.

6. Future work

For future work we are mainly interested in developing more refined techniques for uncertainty treatment. For example, certainty degree values can be combined on the basis Necessity/Possibility Theory [1] which provides a way of reinforcing certainty values in polygon intersections. In case we need more accurate maps because polygons cover bigger areas than the real occupied ones, it is also possible to add information about free space. If a ground robot sensed an obstacle it is because it could be placed somewhere nearby. This information might be useful to determine a path avoiding that obstacle by including a description of the area where the robot was placed into the corresponding obstacle area –which must have sonar polygons. In this way, free space areas bound sonar-detected obstacles. In addition to that, free space areas can also be added to those obstacle areas with low certainty value polygons that were actually crossed by ground robots, so that they force certainty values to be decreased and thus the map to be corrected.

Acknowledgments

We thank Enric Celaya from the Industrial Robotics Institute (IRI-CSIC at Barcelona, Spain) to allow Maite López-Sánchez to apply to this work what she learnt about outdoor research from her current work in the NEMO Project. We also thank Damian Hesse for working on the image processing and Gogksel Dedeoglu for grouping sonar readings in the ground robots.

This work is supported in part by DARPA under contract F04701-97-C-0021.

Maite López-Sánchez is a visitor supported by the Spanish Research Council (CSIC) and by the Catalan Research Council (CUR).

References

[1] Dubois and Prade, “Possibility Theory”, Plenum Press, New York, 1988.

[2] Kortenkamp D.M., “Cognitive maps for mobile robots: a representation for mapping and navigation”, PhD dissertation, CS dept. University of Michigan, 1993.

[3] Lim and Cho, “Physically based sensor modeling for sonar map in specular environment” in Proceedings of IEEE International Conference on Robotics and Automation, pp. 1714-1719, 1992.

[4] Lopez-Sanchez, Lopez de Mantaras and Sierra “Incremental map generation by low cost robots based on Possibility/Necessity grids” in Proceedings of Uncertainty in Artificial Intelligence, pp. 351-357, 1997.

[5] Movarec and Elfes, “High resolution maps from wide angle sonar” in Proceedings of IEEE International Conference on Robotics and Automation, pp. 116-121, 1985.

[6] Pagac, Nebot and Durrant-White, “An evidential approach to probabilistic map building” in Proceedings of IEEE International Conference on Robotics and Automation, pp. 745-750, 1996.

[7] Pearl, “Heuristics: Intelligent search strategies for computer problem solving ” Addison-Wesley Publishing Company, 1984.

[8]Tony J. Prescott, “Spatial representation for navigation in animats” in Adaptive Behaviour 4(2), 1996.

[9] Russel and Norving “Artificial Intelligence, a modern approach” Prentice Hall Editors. Chapter 25: Robotics. 1995.

[10] Tod S. Levitt and Daryl T. Lawton, “Qualitative navigation for mobile robots” in Artificial Intelligence 44, pp. 305-360, 1990

[11] Werger and Mataric “Robotic Food Chains: externalization of State and program for minimal agent foraging” in Proceedings of the 4th international Conference on Simulation of Adaptive Behavior, 1996.