Robotic object searching strategies based on partial knowledge

(Bachelor Project)

Noël Lüneburg, s1773135, n.luneburg@student.rug.nl,
Tijn van der Zant* and Amirhossein Shantia*

September 12, 2013

Abstract

A robot assisting at home should be capable of searching for objects with incomplete knowledge of where these objects might be. The knowledge that it does have is the chance that an object will occur on any piece of furniture in the house. In this research, two searching strategies will be compared, through use of a performance measure consisting of multiple dependent variables. To test the searching implementations, the team BORG's Robocup@Home robot was used to attempt the task of finding certain objects when given a map with annotated furniture. The experiment gives us an insight into one example of how an autonomous robot can act with partial knowledge. The results show that both strategies perform similarly, even though errors that appear during searching seem to be caused by different factors depending on the searching strategy.

1 Introduction

When robots are tasked with finding objects in an environment it would not be a great challenge if the robot knows a priori exactly where these objects are located in addition to the areas it can walk across freely. However, often in object searching situations, there will be at least some uncertainty about either obstacles or the locations of the target objects themselves.

The more specific problem introduced in this paper occurs when the robot is tasked with the exploration of a given area for which an annotated map is provided. During this exploration, one or multiple target objects need to be found. The 2D map contains information about obstacles in a trivial way, such as walls and static obstacles. The annotations describe obstacles or locations at which there is some chance for these objects to be positioned. These annotations are virtual and accessible for the robot from its memory as opposed to physical annotations that are placed in the real world (such as landmark tags used in [1]. Annotations are not only useful to add information to a map. The view that it enhances human-robot interaction is also supported, in the way that it simplifies commands [2].

If the annotations consisted of a single coordinate in 2D-space, a well known searching strategy such as A* [3] could be used to plan a path that includes each location which possibly contains an object and execute an object detection algorithm at each of these locations. However, annotations of a single point are not realistic, as objects are often found on for example tables, shelves or couches. Each of these locations have a shape and size and an object can be positioned anywhere in this region. Depending on the size and shape of these locations, the robot might have to search from multiple perspectives around some locations to cover the whole region. This paper introduces a method to search an area with an annotated map for specific target objects, where the annotations are represented as polygons.

The motivation for such a searching method

*University of Groningen, Department of Artificial Intelligence: Cognitive Robotics Laboratory is that it will be a general searching method, independent of many properties of the locations that hold target objects. As long as a map is provided and the annotations have a property that states the approximate chance of occurrence of each object, then the method will make sure the robot navigates to every necessary location to be checked for objects until every location is checked or until the robot is commanded to stop searching.

The experiments described in section 2.6 are constructed in an exploratory fashion. Two main searching strategies (section 2) are compared to see which algorithm finds the most objects and searches the quickest.

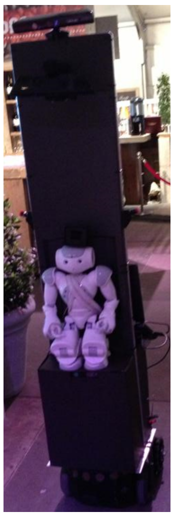
Figure 1: Photo of the domestic service robot, called "Sudo".

The type of robot used in the experimentation of object searching is a domestic service robot (see figure 1). The robot is capable of navigation, object detection and obstacle avoidance among other features. For more details regarding its modules, see section 2 below.

Just like the robot itself, the motivation for the experiments in this paper come from the applications in domestic areas, mainly focused on healthcare. A robot operating autonomously in a domestic environment should be able to find specific objects without being given the exact locations, removing the need for human participation in the task.

2 Method

2.1 Task

Given a map annotated with potential object locations, the robot must approach each location and fully search it for objects. A finite set of objects is used. The search finishes when all locations have been searched. Searching does not terminate when all objects have been found due to unreliability of the object detector module. This will be explained below in more detail.

2.2 Performance measures

There are two factors that define the performance of an executed search: **duration** and **number of objects found**. There is a trade-off between these, as the duration can be decreased by making the search at each location less refined. This increases the chance of failing to detect an object. Combined, these two measuring factors give an indication of the searching performance.

There is a chance that the robot failed to detect an object in its field of view due to an error in the object detection module. For this reason, object detections are also tracked manually as the actual detection module is not in the scope of this experiment. Also for this reason, the robot will always complete a search of an area, even when it thinks it has found all objects.

2.3 Modules

To fully complete the searching task, the robot employs multiple modules:

- Object detection module: Uses the feed from a webcam that is mounted on the front of the robot to detect objects. The detection algorithm used is SURF [4]. The detection takes place solely when the robot is standing still, with a fixed camera orientation. This is to place emphasis on the robot's position rather than taking advantage of methods such as predicting at what position there are higher chances of detecting an object [5] or changing the camera's orientation.

- Obstacle avoidance module: Uses an Xbox Kinect [6] to detect obstructions in front of the robot and sends this information to the navigation module.
- Navigation module: Uses the `move_base` [7] package from ROS[a] to plan a path to a given goal using the information from the map and the sensors.
- Localisation module: Uses the Adaptive Monte Carlo Localisation algorithm [8] to find the robot's position given sensor data and a two-dimensional map.
- Approach surface module: Uses the feed from the Kinect to segment surface planes from the robot's forward view. It can also align the robot to the detected surface and move to a specific distance from the surface.

2.4 Annotation process

To place annotations on a map with necessary information, a tool was developed. Annotations are placed on a map completely manually to guarantee their accuracy. In certain environments or for example when there are too many needed annotations, an automatic method could be used, such as [9]. Automatic annotations are more suited to be applied in specific environments that include clear and predictable objects. The tool lets a user annotate regions on a two-dimensional map of the area to be searched. Such a region is represented by a polygon of any shape, allowing any type of location. A name for the location can be specified so that one could for example say "Go to large kitchen table" as a command, instead of coordinates. The height of a location can also be specified. This is not used in the experiment discussed here but it is important to include in the robot's knowledge, because it will need to adjust the camera's pitch depending on the surface height due to the limited field of view. Lastly, partial knowledge of each object is supplied, in the form of probability categories. These categories are: *guaranteed*, *likely*, *unlikely* and *impossible*. These probability groups indicate the chances of each object occurring on the annotation. This information is useful to estimate which locations are more valuable to visit.

[a]Robot Operating System

2.5 Implementation

The information that was stored in the annotation tool can be used to construct a searching behaviour. The first issue is to decide in which order the robot should visit each location. The robot has a starting point somewhere in the area and needs to figure out which annotation to visit first. To accomplish this, a cost function is computed. It takes into account the distance to each annotation, as locations that are close by should have a higher priority to reduce travel time. It also takes into account the chance of object occurrences. The robot might experience a time limit for an area search, so therefore it should attempt to search at places with a high chance of objects first. These factors are calculated by equation 2.1. A cost value for each location is computed and the highest value is selected as the next target location for searching.

$$cost_i = prob_i - d(current, pos_i) \tag{2.1}$$

Where $cost_i$ is the cost-value for location i, $prob_i$ is the summed probability of objects for location i, $current$ is the robot's current position and pos_i is a point near location i. The function $d()$ computes the Euclidean distance between two points, as shown in equation 2.2.

$$d(P, Q) = \sqrt{(P_x - Q_x)^2 + (P_y - Q_y)^2} \tag{2.2}$$

The cost function requires a position near each location. The most logical position would be close to the location while also as close as possible to the robot's location. Therefore, multiple points are generated near a specific location and the closest one to the robot is picked as the target location for that location.

To generate points near an annotation, a method to compute the distance between a point and a polygon are used. The pseudo-code below (algorithm 2.1 explains it. The method takes in a minimum threshold ($thresh_{min}$) and a maximum threshold ($thresh_{max}$). These represent the desired distance interval in which points should be generated.

Algorithm 2.1 Distance from point (p) to polygon

```
for every line segment in the polygon (v,w) do
  Calculate the t-value using equation 2.3
  if $t < 0$ then
    Calculate distance between p en v
  else if $t > 1$ then
    Calculate distance between p en w
  else
    Calculate the projection using equation 2.4
    Calculate distance between p en projection
  end if
  if $distance \geq thresh_{min}$ and $distance \leq thresh_{max}$ then

    return TRUE
  else

    return FALSE
  end if
end for
```

$$t = \frac{(\mathbf{p} - \mathbf{v}) \cdot (\mathbf{w} - \mathbf{v})}{||\mathbf{v} - \mathbf{w}||^2} \tag{2.3}$$

$$projection = \mathbf{v} + \mathbf{t} * (\mathbf{w} - \mathbf{v}) \tag{2.4}$$

The cost function is updated each time the robot finishes searching a location, to compute the next best location to visit as it goes along.

When the robot reaches a location, it will execute a search around it, in order to find any objects that are placed there. The two methods to search around a location are explained below.

For both of these methods, the center of a location needs to be computed. There are methods to numerically calculate it [10], but here it was decided to approximate the center. Because we are talking about a location represented by a polygon, the goal is to find the centroid. This is done by making a bounding rectangle that encloses the polygon. A few hundred points are randomly created within this bounding rectangle and the points that lie within the polygon are kept and averaged to find the centroid. In this experiment the method used to determine whether a point lies within a polygon or not is the horizontal ray trace method (or the even-odd method) [11]. There is another –possibly more efficient– method, but it has limitations, such as the requirement for the polygon to be convex [12].

Step-by-step method

This method is based mainly on the orientation of the robot. When it reaches a location it computes the angle towards the center of that location and then faces it. The robot then moves towards the surface of the location and approaches it up to a specific distance (set at 30cm). Now, it tries to find an object in its view with the use of the object detector module.

Once the robot has detected an object from the set of target objects, or when it has not detected anything for 6 seconds, it continues. The next action is a rotation of 90 degrees to the right. Assuming the robot was standing perpendicular to the location's surface, it would now be facing parallel. It then drives forward 100cm and faces the center again and repeats the sequence. The approaching module is run each time so that the robot does not wander off during the search.

There is no guarantee that a robot can freely and fully navigate around a location. If the obstacle avoidance module detects an obstruction in its path, the robot will go back to the starting point of the location and repeat the steps, only now it turns 90 degrees to the left each time. This does not guarantee that the location will be fully covered, but the fallback should be sufficient to cover most of it. The algorithm terminates when the robot has been sent to the starting position for the second time and also when it comes within two meters of the starting position after having stopped at least four times to detect. This would indicate the robot has done a full lap around the location. The position counter is reset when the robot goes back to the starting position.

Pre-computed points method

The method discussed here uses generated points to decide at which points to search at a location.

Several random points (eight in this case) are generated near the location and these are sorted by angle, so that the robot can traverse them in clockwise or counter clockwise order. At each point, the same steps are taken as with the step-by-step method; the robot faces the center, approaches the location and tries to detect an object. Between these points, the navigation module is used to send the robot to the next generate point. If the next point cannot be reached it will be skipped.

The drawback with this method is the number of searching points used. When this number is relatively low, they might not be spread out around the full area of the location and if the number is too high, then the search would take a long time, as the robot would stop at each of these points to attempt detection.

2.6 Experimental setup

The setup consists of two parts. In the first part, the two searching strategies are tested, by taking one location with varying objects that are placed at it. The second part is run in a simulation of the specific area, where different versions of the cost function are tested. By making this split between experimental setups, the results are more accurate, since there would be more dependent variables if the two were to be combined.

Searching implementations

A single, large table was placed in a hallway while leaving enough room for the robot to move around. Each of the two strategies described above were tested by letting the robot execute six searches for objects. Within these six, the number of objects and their position of placement on the table was varied. However, both algorithms were presented the same six scenarios. In every scenario, the robot was started at a random location near the table.

During each of the searches, the total execution time was tracked, as well as the number of objects that had been detected. Any missed detections due to errors in the object detector module were manually corrected.

Cost functions

To test multiple versions of the cost function, a map of an arena from the latest world championship Robocup@Home [13] was used. The arena contains multiple rooms and multiple locations were annotated on the map: bar, kitchen table, living room table, bed and side table.

The three cost functions used are the standard one (equation 2.1), named *normal*, a *weighted* function (equation 2.5) and a *quadratic* function that uses the normal function with adjusted object chance values, according to equation 2.6.

$$cost_i = 0.6 * prob_i - d(current, pos_i) \tag{2.5}$$

$$value_o = (\frac{chance_o}{4})^2 * 4 \tag{2.6}$$

Where $value_o$ is the value for an object o, and $chance_0$ is the chance of occurrence of object o as a category value c, where c is an element of the set: {0,1,2,3}.

The motivation behind the *weighted* function is that the navigation of the robot is fairly slow in comparison to the actual searching. By lowering the effect of the object values on the cost function, the distance to a location becomes more important and therefore closer locations will be more often selected to visit first.

The *quadratic* function converts the linear property of object probabilities to a quadratic relation. This means that an unlikely chance for an object to occur (a value of 1) will not be twice as small as a likely chance for an object to occur (a value of 2). Instead, the higher the chance, the more effect it has on the cost function. The motivation for this version of the function is that a robot should be more certain about the occurrence of objects before deciding to navigate to and search a whole location that might have unlikely chances of objects.

Each cost function is tested three times on the same map, while varying the object chances and locations to test multiple configurations of a scene. Each cost function is presented the same three configurations.

The tests were done in a simulation program this time. The simulation accurately simulates the robot's sensor information and the world around the robot. However, not all sensors are accessible in simulation and therefore the searching procedure around a location once the robot reaches it is skipped. This does not matter for the results, as the only goal here is to test navigation between locations. Logically, the time of the full navigation procedure is recorded as a measure of performance. Of course, in a real situation, the robot would search each location for objects, so when a robot visits a location, every object that was placed at this location in the current configuration is considered found. Therefore, once all locations holding objects have been visited, the navigation time is stopped. There is no use to continue visiting the other locations because there cannot be any more objects.

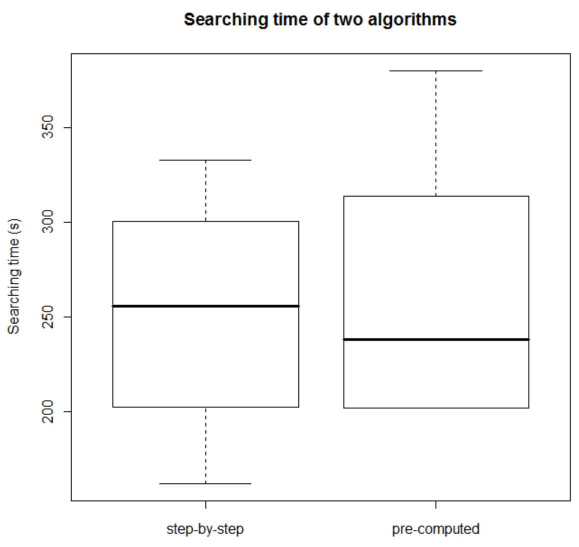

Figure 2: The searching duration of both algorithms.

3 Results

To evaluate the performance of the two searching strategies described in the previous section, one has to look both at the duration and the number of objects found. These cannot really be combined into one performance measure because it is not known how important each factor is. This is dependent on the application. Regarding a specific application, knowledge of the valuation of performance measures becomes available, such as a requirement for a fast searching due to many locations, or a requirement for accurate searching because of important objects. For this reason, the performance of the two algorithms is evaluated independently for both factors.

A box plot of the total search times for each algorithm is shown in figure 2. There is no significant difference between them. The same holds for the comparison of number of objects found, as shown in figure 3. Neither strategy was able to find more than two objects during a search and they show no significant difference.

Box plots that show the navigation duration for each applied version of the cost function are shown in figure 4. All versions show scientifically equal performance in their results, as the differences are not significant.

4 Discussion

Even though there were no statistical differences between the measured performance values, there were some behavioural differences.

The *step-by-step* often veered away from the table. The main reason for this was a faulty orientation at one of the searching points near the table. An erroneous point is computed as the next target position and in this way the error remains in the system. It is true that the robot will return to its starting position if it computed a point that could not be reached, however, it only does this once during a search and on the next occurrence it terminates the search, leaving a potential area of the location uncovered. This was the main reason for failing to detect objects when using the step-by-step method.

An other reason why objects were not found was because of localisation and navigational errors.

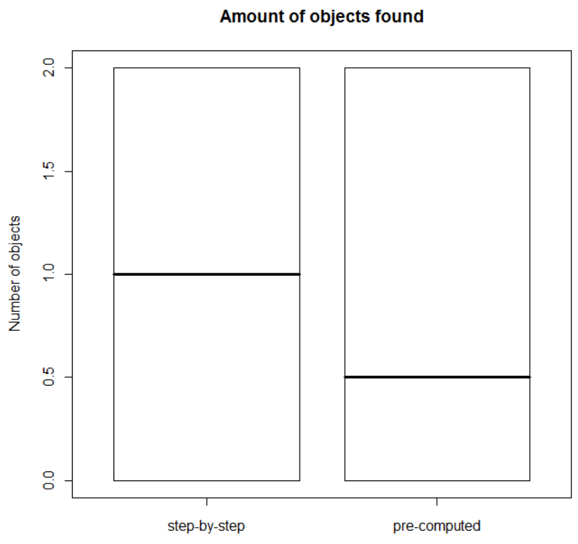

Figure 3: The number of objects that were found by each algorithm during a search.

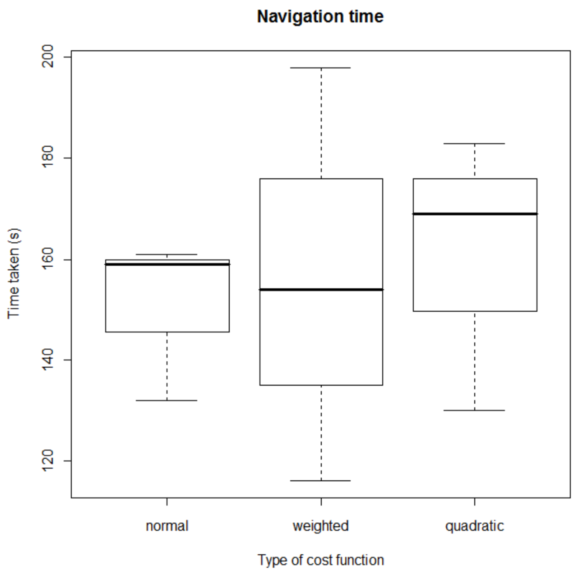

Figure 4: The total navigation time when different cost functions were applied.

The *pre-computed points* method heavily relies on localisation to work properly. However, often when using this strategy, the navigation module lost the robot's current position and a short localisation behaviour was automatically run to recover the current location. Not only does this take time, increasing the total searching duration, the current position also sometimes got offset in comparison to the robot's actual position. This lead to failing to reach target positions and therefore to skipping searching points. More searching points are better because then there is more chance for any object to be in the view of the camera.

The two searching algorithms differ substantially in implementation, so one could have expected a difference in performance. More tests and location configurations may be required to find significant differences. Improving the accuracy of the localisation and navigation module would make results more consistent.

The lack of differences between the application of different cost functions in the duration of simulated navigation might be explained by the vicinity of the locations to each other. When multiple locations are close to each other, the visiting order does not have a great effect on the total time taken. However, in a vast space with multiple locations the distance factor in the cost function would be very large, which would mean that the object values factor would not affect the order of visiting locations because it is much smaller than the distance factor. If, however, a cost function was used with a major emphasis on object values, it would lead to significantly different results from alternative cost functions, because the robot would not mind navigating long distances to get to the 'richer' locations first.

Testing the cost function must be done in many situations, due to the nature of the function. Therefore, many test runs must be done when comparing multiple cost functions. If there were more tests in this research, then potentially, differences between cost functions would show.

5 Acknowledgements

I would like to thank dr. Tijn van der Zant, Amirhossein Shantia and the rest of the robocup team 'BORG' for teaching me the knowledge to

operate the robot 'sudo' and for allowing me to take part in two Robocup@Home league competitions. These have been a great experience and we even managed to finish as the fourth finalist in the world championship in The Netherlands in 2013.

References

[1] Xinde Li, Xiulong Zhang, Bo Zhu, and Xianzhong Dai. A visual navigation method of mobile robot using a sketched semantic map. *International journal of advanced robotic systems*, 9, 2012.

[2] Aleix M. Martinez and J. Vitria. Clustering in image space for place recognition and visual annotations for human-robot interaction. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 31(5):669–682, 2001.

[3] Sven Koenig. Minimax real-time heuristic search. *Artificial Intelligence*, 129(12):165–197, 2001.

[4] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer Vision and Image Understanding*, 110(3):346–359, 2008. Similarity Matching in Computer Vision and Multimedia.

[5] Ksenia Shubina and John K. Tsotsos. Visual search for an object in a 3d environment using a mobile robot. *Computer Vision and Image Understanding*, 114(5):535–547, 2010. Special issue on Intelligent Vision Systems.

[6] Microsoft. Xbox kinect. `http://www.xbox.com/kinect`.

[7] Eitan Marder-Eppstein. Move_base path planning module. `http://www.ros.org/wiki/move_base`.

[8] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte carlo localization: Efficient position estimation for mobile robots. *AAAI/IAAI*, 1999:343–349, 1999.

[9] Wei Song, Kyungeun Cho, Kyhyun Um, Chee Sun Won, and Sungdae Sim. Complete scene recovery and terrain classification in textured terrain meshes. *Sensors*, 12(8):11221–11237, 2012.

[10] Paul Bourke. Calculating the area and centroid of a polygon. *Swinburne Univ. of Technology*, 1988.

[11] Kai Hormann and Alexander Agathos. The point in polygon problem for arbitrary polygons. *Computational Geometry*, 20(3):131–144, 2001.

[12] M.S. Milgram. Does a point lie inside a polygon? *Journal of Computational Physics*, 84(1):134–144, 1989.

[13] Thomas Wisspeintner, Tijn van der Zant, Luca Iocchi, and Stefan Schiffer. Robocup@home: Scientific competition and benchmarking for domestic service robots. *Interaction Studies*, 10(3):392 – 426, 2009.