In: German Workshop on Robotics – GWR 2009, June 9-10, Braunschweig/Germany

Probability-based robot search paths

Jan Deiterding and Dominik Henrich

Lehrstuhl für angewandte Informatik III, Universität Bayreuth, D-95440 Bayreuth
{jan.deiterding, dominik.henrich}@uni-bayreuth.de, http://www.ai3.uni-bayreuth.de

Abstract The aim of this work is to show that robots can be improved using knowledge present or gained in previous executions of the task. Here, this knowledge is used to create search paths tailored to the problem at hand. We describe two approaches to create such paths automatically which enable robots to find an unknown position faster than standard paths. The solution presented here is not limited to the robotic domain, but can also be used for other purposes such as searching for injured persons after accidents. Our approach is evaluated through simulations and we show that these paths perform adequately well or even better than standard paths.

Introduction

Industrial robots are able to perform complex tasks without symptoms of fatigue, exhibiting highest precision and speed. However, these tasks are nearly always executed in a fixed environment; that is the precision is gained by ensuring that all objects are placed in exactly the same position every time. All parts need to have the same dimension, position, orientation, etc. Only by employing external sensors such as vision or force/torque sensors, we can enable a robot to deal with imprecisions and variations occurring in the objects and the environment. The price for this flexibility is that sensor based motions are slow compared to pre-computed motions. Especially when searching objects, the time required is significant.

In [4] we have classified changes that can occur between two executions of the same robot program by two characteristics: The origin of the change and the robot's reaction to it (Table 1). Here we differentiate four types of change: (a) An *indeterminacy* is something we are not aware of at this moment, but once we have learned about it, it will remain constant for a prolonged period of time. (b) *Variations* on the other hand occur every time the robot performs the task at hand. (c) *Faults and errors* happen when a sudden, unforeseen change in the workspace occurs. (d) A *drift* is similar to variations but not caused by the task itself, but by gradual changes within the workspace, e.g. the settings of machines and tools change over time.

In this paper, we are interested in ways of dealing with variations. A search motion must be performed to determine the variation in the current execution of the task. The central idea of this paper is to re-use knowledge gained in previous searches to create search paths tailored to the task and thus shorten the time span required for the search.

The rest of this paper is organized as follows: In the next section, we give a short overview of related work concerning this topic. Based on this, we describe how a search path can be optimized with regard to a given probability density. Additionally, we introduce a modification to our sorting algorithm to further optimize the path for probability densities with multiple maxima. Then we show the validity of our approach in simulations and compare the results with standard search paths in a two-dimensional environment.

		Origin of change	
		Caused by the task	*Caused by abrasion*
Reaction to change	*One-step learning*	Indeterminacies	Faults and errors
	Continuous learning	Variations	Drifts

Table 1: Classification of changes that can occur between multiple executions of the same program. [4]

Related work

The topic of robots performing some kind of search is very broad, so we will only refer to work dealing with searches in industrial environments and results which can be transferred to this domain.

A good overview is given in [8]. Despite the fact that sensor data processing has made significant progress allowing for relatively fast processing capabilities, standard search motions which only use minimal sensory information are still commonly used in industrial applications. A reason for this is that especially small and medium companies lack experts skilled in sensor data processing. Also of interest are the works of [10] and [6] which cover robot motion planning based on sensor data and probability densities.

If a search cannot be avoided, usually cameras are used that supervise the search area for the given variation. While this approach is straightforward and has the advantage that the localization can be made while the robot performs some other task, this is only applicable if the search area can be monitored at all. A typical example for a task where this is impossible is the assembly of a gear box in a car. Tolerances are extremely small and the search area is occluded by other parts of the vehicle so camera supervision is impossible and local sensors must be used.

Sharma [7] incorporates stochastic models into gross motion planning and defines a stochastic assembly process that yields increased performance.

An important area of research outside the industrial domain is *complete coverage paths* in mobile robotics. Here a robot must cover an area e.g. to search for injured people after an accident. A good overview of this area of research is given by [12]. Also of interest is [11], which uses a genetic algorithm approach and knowledge gained in previous executions to optimize the path of a mobile robot.

In summary, efficient search strategies are one of the central problems of robotics. While there are many specific solutions, e.g. [1] and [5], these are nearly always tailored towards specific tasks and the results can rarely be transferred to other areas. Here, we take a more general approach to search motions for industrial applications and outline the requirements for optimized search strategies. In industrial applications, the search area is precisely defined and does not change over multiple executions. The ideas presented in this paper are independent from the type of sensor used, the only requirement is that it provides a binary decision whether the goal of the search has been found or not. We only deal with the search itself, not with any actions that have to be taken by the robot afterwards, e.g., actually inserting a peg into a hole. Examples how this can be achieved using sensors are given in [2] and [3].

Search paths based on probability densities

In this section, we present the concept of a search path generated along a given probability density. The idea is that the robot stores successful positions from previous executions and creates a probability density from this knowledge. This can be achieved by employing the methods described in [9]. The path is not fixed and may change with every update of the probability density.

Robot search paths

A search is a motion that covers a specified area in order to locate an object whose exact position p_g is unknown. Exactly one object is searched for at a time. Here, we set the following preconditions:

1. The search area can be m-dimensional, but its boundary in every dimension must be a straight line. The area can be divided into a set S of cells describing discrete (hyper) cubes with fixed edge length Δc. When the robot moves to a cell, the whole area covered by that cube is probed. The decision whether p_g is found is binary, so there are no hints guiding us towards the goal. We assume that it takes a constant time span to check if the goal is located in a cell.
2. Any movement between two cells is allowed. There is no need for a neighbouring connection between two cells. No cell lying between the current and the next is tested when moving there.

3. A valid search path $P = (p_0,\dots,p_{|S|})$ must visit each cell of the whole area at least once. We include the possibility that the search fails: $p_g \notin S$. A search path is then an ordered sequence of all cells in S.
4. The distance between two cells $c_i, c_j \in S$ is relevant when planning the path. There is a positive cost function $d(c_i,c_j)$ describing the time and effort to move from cell c_i to c_j. Two neighbouring cells have unit distance.
5. There is a probability density φ describing the chance that the object lies within any given cell. This density may be continuous.

Additionally, a change of direction in the search path may slow down the motion in order to perform the turn along the path. We disregard this factor here.

Conditions 1 to 4 describe the general requirements imposed on a search path. Condition 5 is a new addition describing the knowledge we have about the location of the object that we are searching. This allows us to begin the search in the most probable cell and descend along the density instead of employing a predetermined path.

There are three criteria along which we compare different search paths to each other: Their total (maximum) length l and the expected number of cells visited $E_C(P)$ as well as the expected length of the path $E_L(P)$ for the given probability density $\varphi(p_i)$ and path P:

$$E_C(P) = \sum_{i=1}^{|P|} \varphi(p_i) \cdot i \qquad E_L(P) = \sum_{i=1}^{|P|} \varphi(p_i) \cdot d(p_{i-1}, p_i)$$

A developer faced with the task of designing a search path should consider two aspects. On one hand, it may be useful to limit the total length to its minimal value, so the path is not exceedingly long. On the other hand, if the (average) search time is crucial, it may make more sense to create a search path with higher total length but lower expected values. The decision, which of the two expected values is more important, depends on the type of search: If the movement between two cells is relatively fast compared to the time it takes to check a cell, the number of cells visited is significant. In case of slow motions, e.g. controlled movements along surfaces, the expected length is of more importance.

There are two de facto standard search paths for searches in two-dimensional environments: A zigzag path and a spiral path. Both paths can be easily extended to more than two dimensions. The zigzag path is usually chosen if the probability density is uniform, so there is no need to start in a specific cell. The spiral path is usually chosen when φ is unimodal, e.g. Gaussian, with mean in the middle of S. In this case the search starts in the most likely position and gradually descends along φ. Note, that both paths are optimal regarding their length; no cell is visited twice.

Optimizing search paths

Now, we are interested in finding search paths that optimize the expected values for a given probability density. A search path which is minimal in this sense will find p_g as soon as possible.

To create a search path P with lower expected values than a standard path for the given dimension of S, one has to approach cells with high probability first while neglecting cells with low probability until the end of the search. In case the expected length of the path is of importance, it should be attempted to minimize huge jumps across S as much as possible. The downside is that the distance of two consecutive cells in P now may be much higher than 1. So this search path may not be minimal with respect to the total length.

In a *sorting* strategy to generate an optimized search path, the cells of S are ordered like this: The beginning of the path is the most probable cell, so

$$p_0 = \{c_i \mid c_i \in S \wedge \forall c_j \in S : \varphi(c_i) \geq \varphi(c_j)\}$$

The remaining cells of the path are chosen by a recursive definition: We always choose the next cell according to its probability in relation to its distance d to the current cell, so

$$p_{k+1} = \{c_i \mid c_i \in S \setminus \{p_0, ..., p_k\} \wedge \forall c_j \in S \setminus \{p_0, ..., p_k\} : \frac{\varphi(c_i)}{d(p_k, c_i)^n} > \frac{\varphi(c_j)}{d(p_k, c_j)^n}\}$$

The impact of the distance when choosing the next cell is controlled by the exponent n, which must not be negative. The choice of this parameter depends on the type of application and must be chosen by the developer. The lower the value of n, the lesser the impact on the distance in the selection process. So cells with a high distance to the current cell may be selected as well. The higher n is, the more the selection process favours cells which lie close to the current cell. Note, that if n is set to zero, the cells are simply ordered along their respective probability. This will minimize the expected number of cells visited, but result in an extremely long total path, as it will cover great distances to move from one cell to the next (Figure 1, left). Vice versa, if n is set to infinity, the path always moves to neighbouring cells (Figure 1, right). Technically, we cannot get stuck in dead ends, because of Condition 2. But it is possible that we must move to a cell far away from the current one, because there are no neighbouring cells left. This strategy is not heuristic but always computes the best path for the given probability density and choice of n. It may be possible that more than one path exists with the same expected value. An example is shown in Figure 1 on the right for a Gaussian distribution. All spiral paths that start in the center will have the same expected value regardless of the fact which neighbouring cell is visited first. The strategy presented here only computes one of these paths. Which one this will be depends on the ordering of cells with the same probability and relative distance to the current cell.

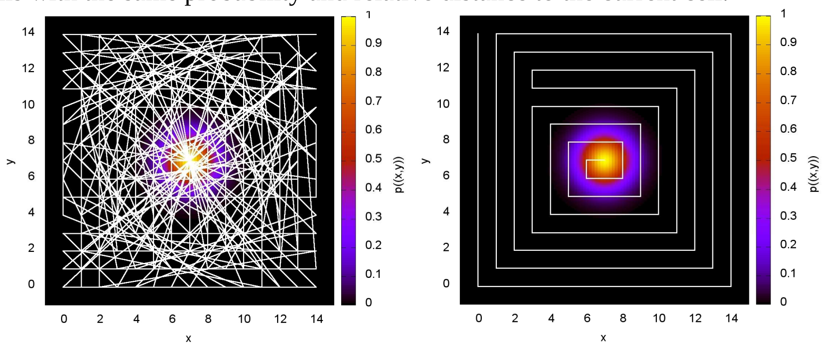

Fig. 1: Impact of the distance when sorting cells. Both paths start at the center. Left: The relative distance between the cells has no impact at all. Right: The relative distance between the cells is of infinite impact.

Search space subdivision

This ordering works very well if φ has only one maximum or if the impact of the distance is chosen so that only cells close to the current cell are selected. But, if there are two or more maxima with a significant distance relative to each other (Figure 2, left), the path generated by this approach will oscillate between these maxima. Because of this, we extend our approach and introduce a *divide-and-conquer* strategy to split S into separate regions. We will repeat this process until the generated regions only contain one or zero maxima and then generate separate, optimized paths for each region with the sorting strategy. Finally, we connect all subpaths once more with the sorting strategy.

The algorithm in pseudo code looks like this:

```
createSmartPath(bool top, list<path> paths) {
      if(findMaxima() == 0) {
            createZigZagPath();
            addThisPathToList(paths);}
      else if(findMaxima() == 1) {
            orderCellsByProbability();
            addThisPathToList(paths);}
      else {
            subspaces = splitSearchSpace();
            foreach(subspace)
                  createSmartPath(false, paths);}
      if(top)
            finalpath = orderPathsByProbability(paths)}
```

The function call in Line 0 takes two parameters: The Boolean parameter `top` describes if this is the topmost function call and the list `paths` is empty initially. Successive calls of this function will add subpaths to this list. In Lines 1 to 6, either a standard zigzag path or a probability ordered path is fitted into the given region, if there is none or only one maximum in that area. The result is added to `paths`. Otherwise the area is split into separate regions (Line 8). A simple approach is to perform a horizontal and a vertical cut through the geometric center of *S*. Other approaches would be to employ Voronoi diagrams between the maxima for a more fitting split. If the area is divided into separate regions, the same function is called to determine a path for this region (Lines 9 and 10). As a result, we have calculated a subpath for every region of the whole area. Now these paths are connected to each other by using the sorting strategy (Lines 11 and 12).

This extension yields the advantage that the path will not oscillate between two or more maxima, but remains in the immediate surrounding of one maximum. The downside is that – regardless of the value of n – there will always be a significant distance we have to bridge between two regions. This will worsen the total length of the path.

Experimental results

In this section, we describe simulation results to show how optimized search paths compare to standard search paths for various probability densities. We have limited the simulations to a two-dimensional workspace. In this case, the paths are already complex compared to a one-dimensional search, but can still be visualized.

Simulation setup

We have set up a two-dimensional squared workspace with an edge length of 15 cm. The position we are trying to find is a hole with a diameter of 1.5 cm. We have set the size of the cells in S to Δc = 1 cm^2. This gives us 225 cells in the search area. So, there are 225! possible search paths, which is already too much for a brute-force computation.

We have created three probability densities:

1. A Gaussian density φ_g with mean p_m = (7,7) cm in the middle of S and σ = 1 cm.
2. A mixed Gaussian density φ_m consisting of four maxima at p_1 = (3, 3) cm, p_2 = (11, 3) cm, p_3 = (3, 11) cm, p_4 = (11, 11) cm and σ =1 cm each (Figure 2, left). A typical example for such a density is a peg-in-hole task on a square plate where the hole is not centered and the plate may be rotated by 90°, 180° or 270°.

3. An off-centered density φ_c with a maximum along the third quadrant in a circle around p_m with radius r = 5 cm (Figure 2, right). A typical example for such a density is a peg-in-hole task where the plate may be rotated by any value between 0° and 90°.

We have not used a uniform density, because no knowledge is present in such a density. To that effect, the expected value of all search paths is identical. The only difference will be in the total length. Because of that it is sufficient to use a standard search path.

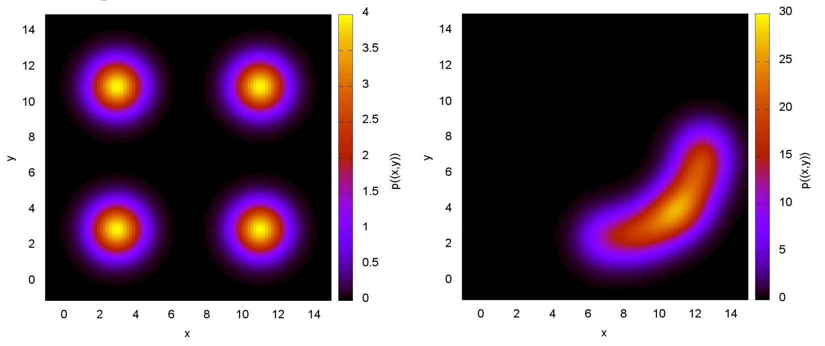

Fig. 2: Multi-modal probability density (left) and density where the maximum is along a quarter circle in the third quadrant of *S* (right) used in the simulations.

Comparison to a standard search path

In two experiments, we used these three densities and generated search paths for each one. The first time, we only used the sorting strategy and the second time the divide-and-conquer strategy. For each density, we generated paths with varying values for *n* in the range [0;10] with increments of 0.1. We then compared the generated paths to a standard spiral path starting in the center and measured the ratio by which the total length and the expected values differ from this standard path. Figure 1 shows two paths for the Gaussian probability density with a low and a high value for *n*. Since there is only one maximum present, both strategies generate the same paths. Figures 3 and 4 show generated paths for a value of n = 5.6 for the second and the third probability density.

We can see that the sorting strategy generates paths that follow the underlying density and prefer neighbouring cells. Nevertheless, these paths can lead to dead ends. In this case substantial jumps have to be made to reach unvisited cells. The divide-and-conquer strategy reduces these jumps by separating *S* into distinct regions. Although here jumps may be necessary as well, these are limited to some extent.

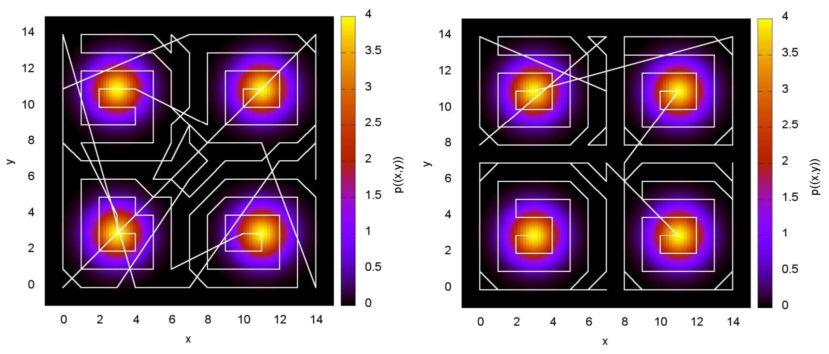

Fig. 3: Generated paths for the multi modal probability density outlined in Fig.2, left. Left: The path generated with the sorting strategy starts at (11,11). Right: The path generated with the divide-and-conquer strategy starts at (3,3).

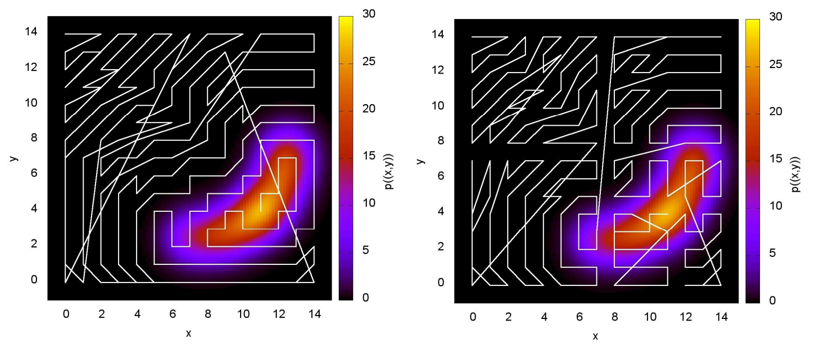

Fig. 4: Generated paths for the quarter circle probability density outlined in Fig. 2, right. Both paths start at (13,4). Left: Path generated with the sorting strategy. Right: Path generated with the divide-and-conquer strategy.

Now we compare both strategies to a spiral path. Figure 5 shows the ratio by how much the generated paths relate to the spiral path for different values of n. We compare all paths by the expected number of cells visited before p_g is found, the length of this expected path and the total path length. All results are set into relation to a spiral path starting at the center of S. A value larger than one means, that the generated path performs superior to the spiral path. Vice versa, a value below one means that the path performs worse than the spiral path.

Once more, the results of the divide-and-conquer strategy for the Gaussian density are not shown, because this strategy generates the same paths as the sorting strategy. In all cases, the total path length (blue and brown lines) is worse than that of the spiral path, which is already optimal. But, with higher values of n, the paths draw near the optimal length. For low values of n, the divide-and-conquer strategy produces slightly better results, because in this case, all jumps are limited to the current region.

For low values of *n*, the sorting strategy generates paths that test significantly fewer cells than the spiral path (red line). But, because these cells are far apart, the expected length covered (green line) can be worse than that of the spiral path. With higher values for *n*, the two lines converge, because now neighbouring cells are preferred, similar to the spiral path. The higher *n* gets, the lower the advantage of the sorting strategy, because of the dominance of neighbouring cells in the selection process.

The divide-and-conquer strategy showed no significant improvement in comparison to the sorting strategy. In case of the off-centered density, the expected number of cells and the expected length of the path (purple and turquoise lines) are higher than the spiral path but lower than the sorting strategy. The overall path length is only slightly better than the sorting strategy and only for low values of *n*. In case of the mixed Gaussian density, the expected number of cells and the expected path length are even worse than for the spiral path, because the splitting algorithm cuts the area and only moves to the next subsection, when the current subsection is completely covered. Because there are many cells with very low probability in every subsection, this lowers the expected values significantly. It may be possible to achieve better results with a more sophisticated splitting algorithm, but this exceeds the scope of this paper.

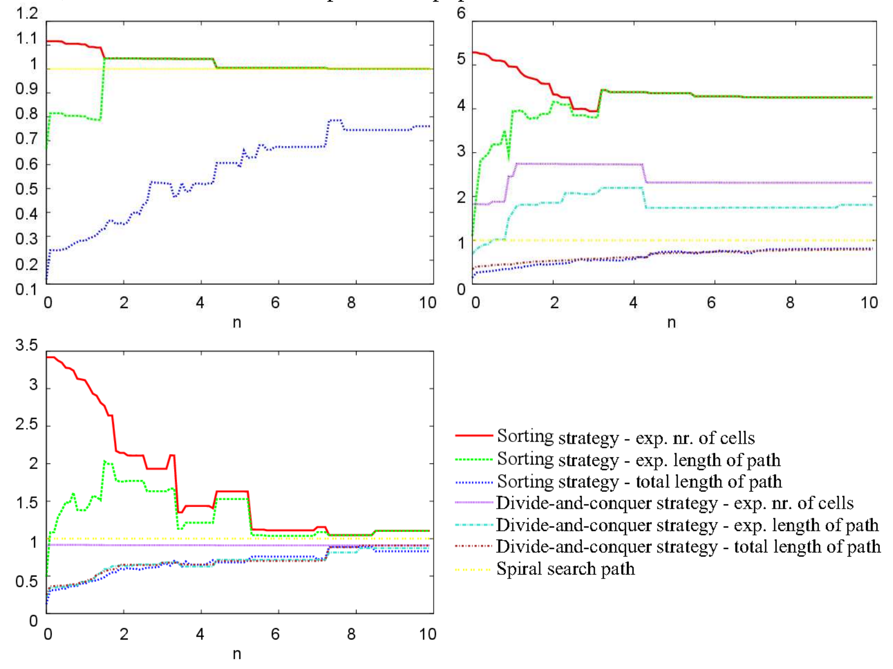

Fig. 5: Comparison of the generated paths to a standard spiral path. Top left: For a Gaussian density φ_g. Bottom left: For a mixed Gaussian density φ_m. Top right: For an off-centered density φ_c.

Random probability distributions

In the next step, we have generated random probability densities in order to evaluate our strategies for a broader set of probability densities. We have created these densities by randomly placing *k* Gaussians uniformly in the search area. For every value of *k*, we have created 100 different probability densities. Then, the sorting strategy was applied to each density with various settings for *n*. We compared the generated paths to the standard spiral path and computed the average for each combination of *k* and *n*. The results are shown in Figure 6.

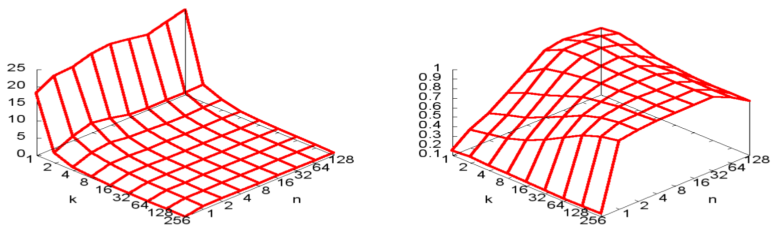

Fig. 6: Expected length of path (left) and total length of path (right) in relation to a standard spiral path for a varying values of *k* and *n*.

We can see that the generated paths perform better than the spiral path for low values of *k* regardless of the choice of *n* in terms of the expected length of the path (Figure 6, left). The more Gaussians are combined the more the overall probability density converges to a uniform density. In this case neither the optimized paths nor the spiral paths are superior because there is no information present in the probability density at all. When we take a look at the overall length of the search path (Figure 6, right), once more we can see that paths generated with a low impact of the relative distance between two cells are significantly longer than the spiral path. With increasing *n*, the optimized paths are nearly as short as the spiral path. There are two noteworthy aspects: The ratio increases faster for high values of *k*. This is because the Gaussians lie closer to each other. But for high values of *k* and *n*, the ratio decreases. This is because now there are so many Gaussians in the overall density that the path tends to get stuck in corners and large jumps have to be made to approach the next free cell increasing the total length.

Conclusion

The aim of this work is to show that search paths based on probability densities are capable of locating the position in question faster than standard search paths. The central idea is to search in areas with high probability of success first in order to maximize the expected value.

We have described the general requirements for path planning and three ways to rate search paths. While standard paths are optimal with respect to the total length, optimized paths can improve these in terms of the average time the search takes. We have shown in simulations that, for Gaussian probability densities, the optimized paths perform almost as well as standard paths and better if there is more information present about the search area. The strategies presented in this paper are no heuristics, but always compute the best path for a given probability density and choice of the impact of the distance between two consecutive cells.

The advantage of our approach is that standard search paths can be seen as special solutions to the more general approach taken here. The algorithms to create optimized paths can be incorporated into the programming environment and no additional knowledge is required by the developer. The update of the probability density describing the search area and the path planner itself can be completely hidden from the developer.

The next step is to test reasonable splitting algorithms for the divide-and-conquer strategy to further improve the expected values. Various approaches to subdivide the search area, such as Voronoi diagrams, can be taken and compared to each other. So far, we have separated the search area into hypercubes. An interesting approach may be to use a hexagonal grid, providing us with more neighbours for a local path planner.

REFERENCES

[1] P. Cheng and D. Cappelleri and B. Gavrea and V. Kumar, ”Planning and Control of Meso-scale Manipulation Tasks with Uncertainties”, Robotics: Science and Systems, 2007

[2] Siddharth R. Chhatpar and Michael S. Branicky, ”Search Strategies for Peg-in-Hole Assemblies with Position Uncertainty”, http://dora.cwru.edu/msb/pubs/iros2001b.pdf, 2001

[3] Siddharth R. Chhatpar, ”Localization for Robotic Assemblies with Position Uncertainty”, 2005

[4] J. Deiterding and D. Henrich, ”Automatic optimization of adaptive robot manipulators”, 2007 IEEE International Conference on Intelligent Robots and Systems, San Diego/USA, 2007

[5] Douglas W. Gage, ”Randomized search strategies with imperfect sensors”, In Proceedings of SPIE Mobile Robots VIII, 1993, pp. 270–279

[6] M. Khatib, ”Sensor-based motion control for mobile robots”, Maher Khatib Sensor-based motion control for mobile robots PHD thesis,LAAS-CNRS December, 1996

[7] R. Sharma, S. LaValle, S. Hutchinson, ”Optimizing robot motion strategies for assembly with stochastic models”, IEEE Trans. on Robotics and Automation, 12(2):160–174, April 1996

[8] Siciliano, Bruno; Khatib, Oussama (Eds.), ”Springer Handbook of Robotics”, Springer Verlag, ISBN 978-3-540-23957-4, 2008

[9] Parzen E., ”On estimation of a probability density function and mode”, 1962, Ann. Math. Stat. 33, pp. 1065-1076

[10] Tenenbein, Aaron; Weldon, Jay-Louise, ”Probability Distributions and Search Schemes”, Information Storage and Retrieval, 10, 7-8, 237-42, 1974

[11] Yanrong Hu, Yang Simon X., ”A knowledge based genetic algorithm for path planning of a mobile robot”, 2004 IEEE International Conference on Robotics and Automation, pp. 4350-4355, 2004

[12] A. Zelinsky, R.A. Jarvis, J.C. Byrne, and S. Yuta. Planning Paths of Complete Coverage of an Unstructured Environment by a Mobile Robot. In Proceedings of International Conference on Advanced Robotics, pages pp533–pp538, Tokyo Japan, November 1993