Robot Motion Planning in Unknown Environments using Neural Networks[1]

Arno J. Knobbe, Joost N. Kok, Mark H. Overmars

Department of Computer Science, Utrecht University, P.O.Box 80.089,
3508 TB Utrecht, The Netherlands, E-mail: markov@cs.ruu.nl

Abstract. We present two approaches to the motion planning problem for car-like robots using an extended Kohonen Self-Organizing Map (SOM). No prior knowledge about the positions of obstacles is assumed. We incrementally build a path from the starting point of the robot towards the goal, using the SOM as a situation-action map. The first approach uses a trial and error strategy to train the SOM. This method is simple but is not always able to escape from dead-end situations. As an improvement a new training-algorithm is proposed that uses edge detection on the visible objects to generate possible motions. Backtracking is used to choose from different possibilities. Experiments show that this new method realizes a considerable increase in performance and speed.

1 Introduction

When building autonomous robot-systems an important problem to solve is the *motion planning problem*: given a robot R and an environment with a set of obstacles, find a path from a given *source* to a given *goal* such that R can move along the path without colliding with obstacles. Many approaches to this problem have been proposed [3]. Recently, neural network techniques have been applied in various ways to solve motion planning and related problems in robotics [1, 4, 5, 6, 7]. In particular, the Kohonen Self-Organizing Map (SOM) [2] has been used to create a map of the free space of an environment [5, 7].

Many known techniques require complete knowledge about position and shape of every obstacle and are therefore not applicable in situations where such prior information is lacking. We describe a new approach to solve the motion planning problem in R^2, with only locally acquired information about obstacles. The only type of sensing we require is that the robot is able to determine the distance to the first obstacle in a number of directions, defining the *visible area*, plus the direction to the goal.

In this paper we consider a special class of robots called *car-like robots* [3]. These are robots that have non-holonomic constraints: they can only move forward or backward while making a curve with a bounded radius, in the same way as real cars. Car-like robots are particularly interesting for our approach because of their restricted set of motions. Our method will only plan forward motions for the robot (backward motions are only used for backtracking). Hence, sensing the visible area in the forward direction is sufficient.

In the simplified case of a point-robot with unlimited motions, the interesting motions can be determined directly by considering local maxima in distances to the first obstacle [6]. With car-like polygonal robots calculating the set of possible motions and extracting a small

[1]This research was partially supported by the ESPRIT III BRA Project 6546 (PROMotion) and by the Dutch Organization for Scientific Research (N.W.O.).

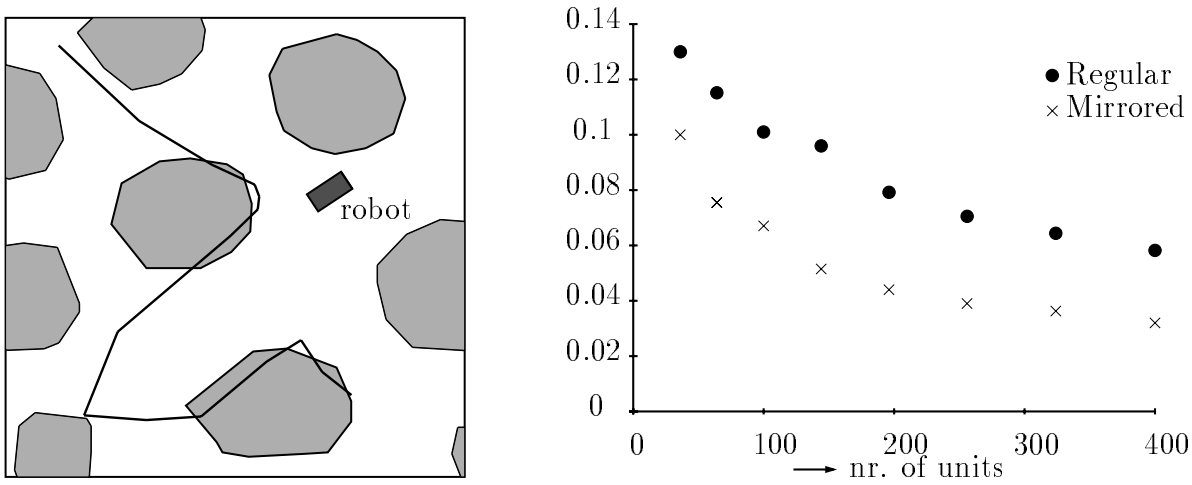

Figure 1: Difference between the actual situation and that of the *bmu*.

number of interesting motions is much harder. A SOM will be trained to combine these two steps in one single computation.

2 Learning Situations

We use the SOM as a self-organizing, adaptive look-up table of situations [1, 4], where each unit u in the SOM has a codebook-vector w_u (input-vector) as well as a datastructure o_u (output-value). The input-vector w_u will represent the situation associated with unit u, described by the distance to the first obstacle in some fixed number of directions and, in the next section, the direction to the goal. The output-value o_u associated with each unit can be anything from one preferred direction to move in, to a list of possible motions (described as arcs with a certain length and curvature). Given the current situation, the *best matching unit* can be determined, and the output-value o_{bmu} now determines the possible actions.

Training of a representative set of situations has been tested in various typical environments. The results of such training in an environment with large convex blocks are illustrated in fig 1. The left picture shows an example situation with the best matching situation stored in the SOM drawn as a thick line. The graph on the right shows the mean squared error between the situation of the *bmu* and the actual situation, as a function of the network-size (•'s). For symmetrical robots, considering the mirrored situations as well reduces the overall error ($\times$'s) as expected.

3 Method 1: Trial and Error

This section describes an on-line, unsupervised training approach to the motion planning problem. It was proposed by Heikkonen et al. [1] as a simple trial and error scheme which learns to avoid obstacles. In the original paper no specific shape or restrictions on motions were assumed. We have tested the approach for a rectangular car-like robot. It turned out that, contrary to the claims made in [1], the method is only successful in simple environments.

The robot moves in an unknown environment, controlled by the SOM. When a collision occurs, the robot will backtrack a fixed number of steps, after which the situations leading to the collision are trained to the SOM, together with corrected actions that will lead the robot away from the obstacle.

The input-vector of the SOM consists of distances in n forward directions, and a direction α towards the goal, where negative values indicate directions to the left of the robot. The output-value consists of just a single value β, the preferred course to take relative to the goal direction. Initially, every unit has an input-vector of n random distances and $\alpha = 0°$. The output-value is just $\beta = 0°$. This means that initially in any situation the robot will try to go straight towards the goal. At every step the robot measures the current situation and uses $\beta_{bmu} + \alpha$ as the preferred direction to move in. It moves a small step forward along an arc turning towards $\beta_{bmu} + \alpha$, thus respecting the non-holonomic constraints.

To detect collisions, the robot is equipped with two detectors on the front, one on either side of the robot. When a collision occurs, the robot will take back m steps. The situation-vectors and associated actions of these m steps are trained to the SOM, with a correction-term $\pm A(k)$, depending on the side and the time till collision, added to the actions.

The algorithm has been implemented, and a number of experiments in different simulated environments were performed. We asume sensory data to be noise-free. A SOM of 10×10 units arranged in a rectangular lattice was used in accordance with [1]. The system was capable of learning simple problems involving convex obstacles with sufficient free space between them. Also, the system could generalise between different environments as long as the encountered obstacles were similar. Typical runs after being trained took approximately 30 seconds.

In more critical situations with not much space however, the accuracy was too low to result in collision-free paths. Apart from the inaccuracy caused by the discrete nature of the SOM, a number of problems, specific to the described approach can be distinguished:

1. There is no way to store alternative actions for use in a backtracking-scheme, because the SOM only provides one solution for each situation.

2. A decision problem will be evaluated several times (possibly leading to contradicting actions), because the robot makes tiny steps. For example, when deciding to pass an obstacle on the left or right, either decision will only be correct as long as the system adheres to it.

3. The training algorithm is incorrect in cases where the robot is faced with a non-convex obstacle.

4. The training of the output-value is incorrect in situations with multiple options. If two similar situations with different actions are trained to the same unit, then the resulting action will probably not be correct. (In fact [1] describes training both the actual and mirrored situation-action pair, which may lead to erroneous results in symmetric situations with asymmetric actions).

5. Learning from mistakes leads to conservative solutions. The robot tends to stay in open spaces, avoiding denser parts of the environment.

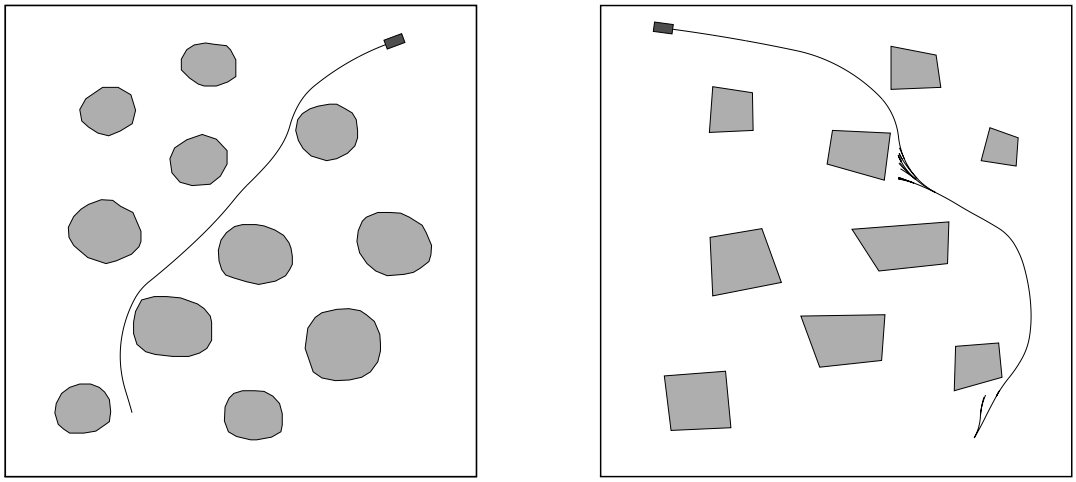

Figure 2: Generalisation from one environment to another.

6. The algorithm is slow, both during training and testing, because only tiny fixed steps are made and only part of the time is spent on training.

It turns out that the method behaves worse than a simple potential field approach without backtracking [3]. Hence, it is not very practical.

4 Method 2: Edge Detection

We propose a new method of assigning the appropriate actions to the set of prototype situations stored in the SOM. The method uses a process of 1-dimensional *edge detection* to generate a set of *edges* in the large set of possible motions. By edges we mean those motions in the full range of possible motions that represent the separation between colliding with an obstacle, and just passing it. The edges provide knowledge about the position of obstacles, and can be used to generate a small set of motions which avoid these obstacles. By finding a set of motions instead of a single motion, we hope to overcome the problems of traditional reflexive methods such as potential field methods [3].

We start by positioning the robot at random configurations in a (simulated) training-environment and use the associated situations to train the SOM, as explained in Section 2. After training we consider each prototype situation w_u and compute a large number of motions in w_u. The robot is moved along arcs with a range of curvatures, and the distance it can travel before colliding is recorded. A set of edges can be generated by identifying big jumps in the distances of travel l_i. We distinguish between positive $(+)$ edges and negative $(-)$ edges, which indicate the left and right side of an obstacle, respectively. The edges can be calculated by first applying a smoothing filter to the distances l_i, and then the actual edge detection-filter:

$$\left.\begin{array}{l} e_i = \frac{1}{4}l_{i-1} + \frac{1}{2}l_i + \frac{1}{4}l_{i+1} \\ f_i = e_{i+1} - e_i \end{array}\right\} \quad f_i = \frac{-l_{i-1} - l_i + l_{i+1} + l_{i+2}}{4}$$

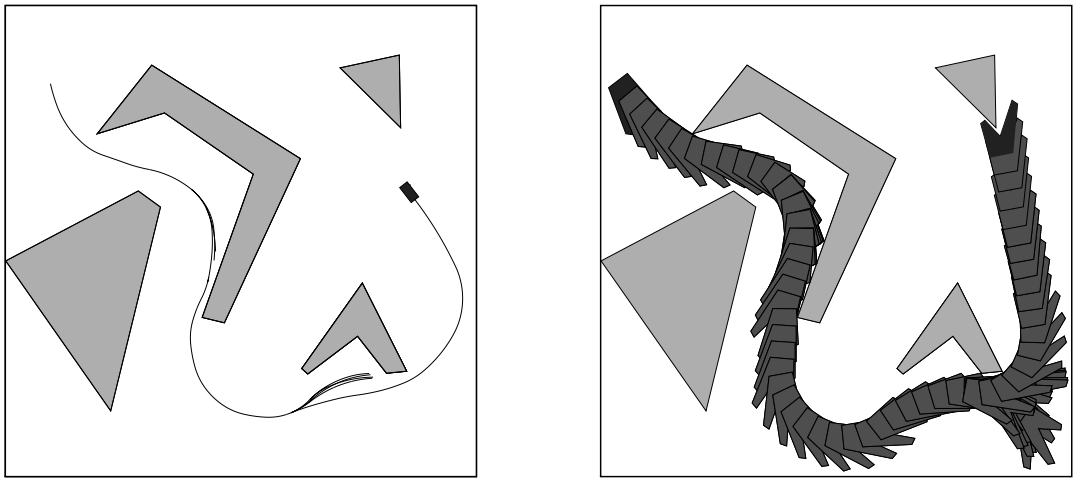

Figure 3: Non-convex obstacles with different robot-shapes.

where e_i are the smoothed values and f_i the final edge-values.

To locate the edges in the list of edge-values f_i, we first distinguish segments in the list that have an $|f_i| > \theta$ where θ is some threshold-value. Of each of these segments the absolute maximum is reported as an edge. For the final set of possible motions we report a motion between every adjacent pair of edges, except for the combinations $(-,+)$ which indicate an obstacle. So for each situation we get a set of a few (typically ≤ 3) possible motions. The notion of reporting motions between edges corresponds to using local maxima as described in [6], only generalised to the case of car-like polygonal robots.

Finding a path is now done by searching the environment in a depth first manner, guided by the SOM. In a given situation we sort the possible motions associated with the bmu with respect to the difference in resulting orientation of the robot and the goal-direction and try the best option first by following (part of) the arc stored. Once the goal is reachable by a direct arc, the search is terminated.

Instead of just using the edge detection method to train a SOM, we can also apply it to guide the robot directly. This will give more accurate results, but computation will become rather inefficient. The SOM-based method can perform faster by doing a form of preprocessing.

The approach has been tested on a number of environments, with a variety of robot sizes and shapes. Some examples can be seen in Figs 2 and 3. Fig 2 on the left shows a training environment for a 20×20 SOM, together with a path computed using the SOM. Fig 2 on the right shows the ability of the SOM to generalise to a different environment. Mirrored situations were included to improve accuracy. Computing runs in these environments took 0.1 seconds on average. Training was rather slow (25 minutes) but we expect that this can be improved considerably (we plan to work on this in the near future).

Fig 3 shows the results on an environment with non-convex obstacles. On the left a SOM of 20×20 units was used, resulting in an average computing time of 0.4 seconds. A larger SOM (30×30) was needed to guide the robot in the environment in fig 3 on the right in 1.0 seconds on average.

The same tests were done using the edge detection method to guide the search in a direct way (without using the SOM). Although the resulting paths were more accurate, the computation-time increased with a factor of more than 70. If a path of a robot is to be planned before actually performing the motion, then the choice of using a trained SOM is clearly more favorable. If planning should be on-line then having to backtrack less often will be preferable because the actual time needed to move the robot will dominate the time needed for planning.

5 Conclusion

We have proposed a new approach to motion planning using neural networks, which improves the algorithm described in [1]. The new algorithm has a number of advantages, most important of which are: backtracking from dead-end situations, a separate training-phase which guarantees correct training, applicability in both on-line planning and off-line planning, and a considerable improvement in efficiency.

Incorporating the difference between the current situation and that of the *bmu* as a calibration, or combining the two methods are promising topics for future research.

References

[1] J. Heikkonen, P. Koikkalainen, E. Oja. From situations to actions: motion behavior learning by self-organization. Proceedings of the ICANN 1993, 262-267.

[2] T. Kohonen. Self-organization and associative memory. Springer, 1989.

[3] J. Latombe. Robot motion planning. Kluwer Academic Publishers, 1991.

[4] H. Ritter, Th. Martinez, K. Schulten. Neural computation and self-organizing maps: an introduction. Addison-Wesley Publishing Comp., 1992.

[5] J. Vleugels, J. Kok, M. Overmars. A Self-Organizing Neural Network for Robot Motion Planning. Proceedings of the ICANN 1993, 281-284.

[6] J. Tani, N. Fukumura. Learning goal-directed sensory-based navigation of a mobile robot. Neural networks, Vol 7, No 3, 553-563, 1994.

[7] U. Zimmer, E. von Puttkamer. Realtime-learning on an autonomous mobile robot with neural networks. Proceedings Euromicro '94.