MODEN: Multi-Robot, Obstacle-Driven Elastic Network Path Planning

Francisco S. Melo
Institute for Systems and Robotics,
Instituto Superior Técnico
Lisboa, Portugal
fmelo@isr.ist.utl.pt

Manuela M. Veloso
School of Computer Science
Carnegie Mellon University,
Pittsburgh, PA, USA
veloso@cs.cmu.edu

***Abstract*— We propose an efficient and elegant heuristic algorithm for path planning that makes use of an *elastic network*. The network initializes as a simple straight path between the initial and goal positions. It then randomly samples the space around the areas of this initial path that lie inside obstacles, "pushing" the network away from the obstacles and thus determining an obstacle-free path. The fundamental algorithm, dubbed as ODEN (Obstacle-Driven Elastic Network), is an improved extension of the path-planning algorithm proposed in [1]. We also propose a multi-robot variation of ODEN, that we have named MODEN (Multi-robot ODEN). We illustrate the application of both algorithms in numerous test environments and discuss its applicability to general path-planning problems.**

I. INTRODUCTION

Given an environment representation, path planning consists of determining the "best" trajectory through the environment linking some initial point to some goal point. The problem of determining such best trajectory is known to be complex to solve, in that general complete algorithms[1] are argued to be computationally too expensive to be of any practical use [2].

As such, many heuristic approaches have been proposed in the literature that aim at optimizing some particular feature of the yielded path or of the path-planning process. For example, a path can be sought to achieve maximum smoothness, using a cost function based on the curvature of the path and its derivative [3]. In that case, the path can be represented as a set of *postures* between which minimum curvature paths are described. Another example may consider the problem of navigating a vehicle from an initial point to a final point with constant velocity [4].

However, most approaches simply aim at establishing a minimal-length, collision-free path from the initial point to the goal point, while making the determination of such path computationally effective.

In many heuristic methods, the use of random search/sampling in configuration space is a commonly used strategy, providing a computationally effective technique of free-space "exploration". For example, rapidly-exploring random trees (RRT) build a tree in the free space by probabilistically balancing the search towards the goal with the exploration of random directions, chosen according to the dynamics of the vehicle [5]. This approach may be further enhanced to cope with replanning problems by also considering past plans in the construction of the tree [6].

RRTs can be included in the broader class of the previously introduced probabilistic path planning algorithms known as *randomized road-maps* [7]. These algorithms randomly sample the configuration space and determine a "road-map" uniting the different sampled points so as to build a path from the initial point to the final point. This road-map is, basically, a connectivity graph linking the various sample points.

Randomized potential fields [8]–[10] is another class of probabilistic path planning algorithms, which define a potential function on configuration space. Similar to randomized road-maps, a path then links local minima from the initial point to the final point. A rather general analysis of randomized methods, namely potential-fields and road-maps, can be found in [2].

There are also several methods relying on quad-tree divisions of the environment, to which search algorithms are then applied, such as the A^* or D^* (see [11]–[13] for details). Finally, we refer to the book by Choset et al. [14] and references therein for a broad treatment or robotic motion planning.

In this paper we present a new algorithm, ODEN, that extends the approach presented in [1]. This method represents a path as an elastic network initialized as a straight-line between the initial and the target points. The different nodes in the network are then "attracted" towards free-space and "repelled" from the obstacles, yielding a collision-free trajectory.

This method has several interesting properties that make it worth of consideration. First of all, as will become apparent along the paper, the method only needs a function which is able to determine whether a point lies in free-space or inside an obstacle. It does not require a complete representation of the environment, since sampling a point in configuration space to determine whether it lies in the free-space can generally be done using sensorial information, require no representation of the complete environment.

Another interesting feature is the fact that the path is represented as an elastic network of units. Even if in this paper we consider path-planning problems for mobile robots, the fact is that other interesting problems can be addressed with this algorithm. As an example, consider that a communication channel is to be established between two distant locations A

[1] An algorithm is complete if it finds at least one solution whenever there is one.

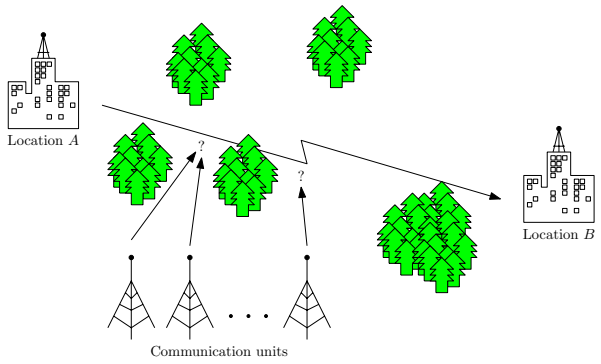


Fig. 1. Communication channel problem.

and B by placing several communication units between A and B (see Fig. 1). Each communication unit will relay the messages from one location to the other by receiving/sending the message packets to the neighboring units. The considered path-planning algorithm is a local method that updates the position of each unit separately by sampling the space around that unit and thus provides a method that allows each unit to individually and gradually adjust its position so that the final network is obstacle-free.

In this paper, we propose a modification of the basic algorithm in [1]. This new algorithm, which we call ODEN for *obstacle-driven elastic network*, retains all the desirable properties of the original method while actually improving its computational efficiency and overall performance. In particular, our method provides less "tortuous" paths and greatly decreases the number of necessary iterations for convergence. We then introduce MODEN, a multi-agent extension of ODEN that determines non-intersecting, collision-free paths for multiple robots navigating in a common environment.

II. PATH PLANNING USING AN ELASTIC NETWORK

The algorithm described in [1] makes use of an elastic network of "processing units" (PU) linking the initial point in configuration space with the target point. The elastic network is initialized as a straight line between the initial and final points. We denote by $\mathcal{U}^t$ the set of all units in the network at iteration t. The algorithm proceeds at each iteration t by choosing a random unit $w_i \in \mathcal{U}^t$ and also randomly sampling a point in a region of radius r around w_i. Once a sample x is obtained, the algorithm updates the position of the nearest node in the network (the best matching unit, BMU) according to the following heuristic:

- If the sample point is in free space, it "attracts" the BMU towards it (Figure 2.a);
- If the sample point is in the occupied space, it "repels" the BMU away from it;
- If both the sample point and the BMU lie in occupied space, the BMU will move randomly, orthogonally to the line segment whose extreme points are BMU's neighboring units (Figure 2.b).

Once the BMU is updated, its immediate neighbors are updated accordingly.

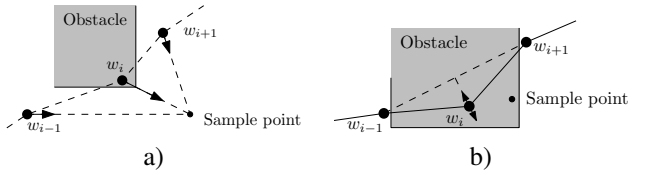


Fig. 2. Several updates of the closest PU, depending on the position of the sample point.

As stated, the points sampled from configuration space used in the updates are randomly obtained by sampling around each of the nodes in the network in a region with radius r. This radius is decreased, as the algorithm proceeds, from an initial value of r_I down to a final minimum value of r_F. This "two-scale" sampling can easily be justified. The algorithm starts by "pulling the network" towards large regions of free space and then, locally, adjusts the network to the "details" of the environment.

Finally, every λ time instants, where λ is a fixed parameter, a new unit is added in between the two most distant units in the network until a maximum number of units is reached.

The term *elastic network* arises from the fact that each unit exerts some conservative tension on its neighboring units so as to maintain their relative positions. In other words, the updated PU is "pulled" by its neighbors and also "pulls" its neighboring units. More details on the algorithm can be found in the referred work [1].

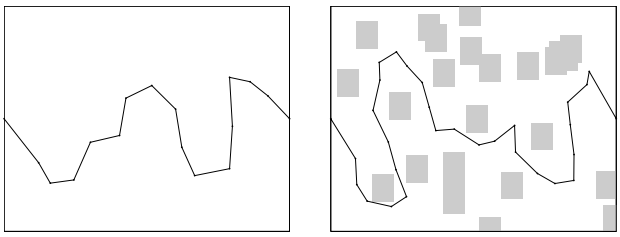
The method has several important advantages. First of all, it is extremely simple to implement. It only requires a sampling mechanism around the network and the evaluation of the attraction function. This attraction function basically determines whether a point is in free-space or in occupied space. The update of each unit then resumes to a few extremely simple operations.

Secondly, it is an extremely efficient algorithm, being in fact able to find collision-free paths between the desired points with very little computational effort. The examples reported in [1] were able to determine collision-free paths in several environments in few thousand iterations. Furthermore, it does not requires a complete model of the environment but just a sampling function able to determine whether a point lies in free-space or not.

Finally, the method is *local* in that each unit is updated considering only its immediate neighborhood. This is a very interesting feature of the algorithm: even if each unit is an independent entity that must adjust its individual position communicating only with its neighboring units and sampling the surrounding space, the method provides a way of determining a *global* obstacle-free trajectory/network.

However, it also presents several inconveniences. First of all, the algorithm will often spend a lot of iterations updating units that need not be updated. This happens, for example, in an obstacle-free environment or any environment where there is a lot of free space, where most of the units will already start away from any obstacle.

Furthermore, because sample points in free-space keep attracting the network to free-space, this may lead to bizarre trajectories such as the ones depicted in Figure 3.



a) No obstacles; b) 25 obstacles.

Fig. 3. Trajectories obtained by the original algorithm.

Finally, the algorithm was designed to run for a fixed number of iterations, usually large, and takes generally this number of iterations to terminate before a path is produced. In the next section we propose a modification of the algorithm which, while maintaining the simplicity of the original algorithm and its functional principle, alleviates the reported inconveniences.

III. THE ODEN ALGORITHM

In this section we describe two fundamental changes to the algorithm in the previous section: a new update mechanism for the units and an additional condition to introduce new units in the network. These modifications yield the ODEN[2] algorithm and readily overcome the inconveniences pointed out to the method from [1]. We also present the results obtained with ODEN that show the introduced modifications to actually improve the overall efficiency and performance over the original algorithm.

A. The update mechanism

Recall that the two main inconveniences reported in the previous section are the many iterations spent updating units that need not be updated and the fact that the *whole* network is attracted towards free-space, this leading to undesirable trajectories.

It should be clear that the undesirable behavior featured is mainly due to continuous updating of units that are already in free space. To overcome such phenomenon we introduce a simple modification to the algorithm: we partition the set $\mathcal{U}^t$ into two sets, $\mathcal{U}_U^t$ and U_F^t. The set $\mathcal{U}_U^t$ contains the "updatable" units and the set $\mathcal{U}_F^t$ contains the "fixed" units. In a first approach, we consider the set U_F^t as the subset of $\mathcal{U}^t$ lying in free-space. Therefore, we update only the units that lie inside objects.

There is however a purpose in continuously updating the different PUs even when they are in free space. Consider the situation depicted in Figure 4. In the original algorithm, the two units depicted would be continuously updated, pulling their common edge out of the obstacle. If we consider the set $\mathcal{U}_F^t$ as containing all units in free space, none of the two units depicted will be further updated and the path thus obtained will not be collision-free, even though all units may eventually lie in free space. To overcome this situation, we consider a second class of updatable units: those whose contiguous edges intersect any of the obstacles. In Figure 5 we illustrate the two classes of updatable units.

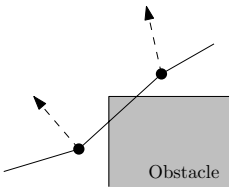


Fig. 4. Situation where further updating is desirable.

[2]Obstacle-Driven Elastic Network.

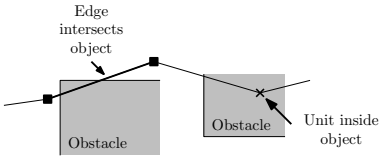


Fig. 5. Two different classes of updatable units.

One final remark to refer that, in order to determine if a given edge intersects any object, we sample several random points along the edge and test whether they lie in free space. The number of points depends on the actual length of the edge. This method generally produces quite reliable results even without sampling too many points.

B. The introduction of new units

Recall that in the original algorithm a new unit is added to the network every λ iterations up to a maximum predetermined number of units. The addition of such extra units, as well as a decrease in the sampling radius around the network, allows the network to converge to smoother trajectories and forces the updates to consider increasingly *local* data around each unit (see [1] for complete details).

Considering the modified version of the algorithm described insofar, it is possible and indeed likely that all units reach free-space before λ iterations have occurred. This implies that the algorithm will "stall" until a new unit is introduced. As such, we include an additional condition in the algorithm, and a new unit is added to the network if any of two conditions is verified:

- Whenever λ iterations have occurred; or
- Whenever $\mathcal{U}_U^t$ is empty.

If the algorithm introduces a new unit in the network before λ iterations have occurred, the sampling radius is, nevertheless, decreased accordingly.[3]

We remark that the introduction of new units also alleviates the problem described in Figure 4. In fact, in many situations such as the one in Figure 4, a new point will actually be added between the two points in Figure 4, leading to a solution like the one on Figure 6.

C. The ODEN algorithm

We refer to the modified version of the algorithm as ODEN, standing for obstacle-driven elastic network. Table I presents the pseudo-code for the ODEN algorithm.

[3]This is implemented by keeping a counter on the number of iterations occurred since the last insertion. Whenever all units reach free-space, the counter skips to the next insertion step. Since the radius used to sample depends on this counter, it is decreased accordingly.

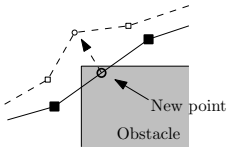


Fig. 6. The extra point solves the problem.

F is the attraction function given by

$$F(x) = \begin{cases} 1 & \text{if } x \in X_{free}; \\ -1 & \text{if } x \in X_{obs}, \end{cases}$$

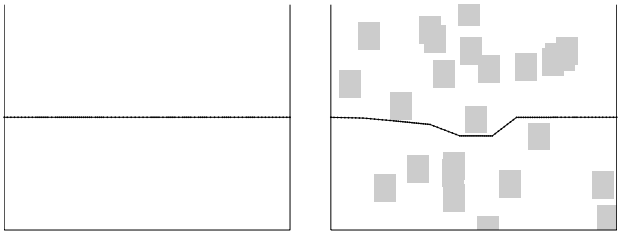
where x is a point in configuration space, X_{free} is the free space and X_{obs} is the space occupied by the obstacles. Clearly, if X is the complete configuration space, $X = X_{free} \cup X_{obs}$.

The parameters $t_{\max}$, λ, β, η_0 and η_1 are common to the algorithm in [1]. The first two parameters represent, respectively, the maximum number of iterations for the algorithm and the number of iterations between two insertions of a new point. The last three parameters basically define the "update rates" and "elastic coefficient" used in the updates of the various components of the algorithm. The input parameters x_I and x_F represent the initial and final point of the path and N_0 and $N_{\max}$ represent the initial and final number of nodes in the network.

D. Experimental results from the ODEN algorithm

We now present the results from several tests, obtained with the ODEN algorithm. In all results displayed, the algorithm was run with the same parameters as those reported in [1].[4]

Figures 7.a and 7.b present the results obtained in the same environments as those in Figure 3. Notice that ODEN is able to overcome the inconveniences of the original algorithm reported in the previous section.



a) No obstacles; b) 25 obstacles.

Fig. 7. Trajectories obtained with the modified algorithm in the environments of Figure 3.

Figures 8 and 9 present the results obtained in two sets of environments with 20 and 40 random obstacles, respectively. Notice that even if the algorithm does not take into account the minimization of the length of the path, the obtained paths often exhibit small perturbations from the initial straight path.

[4]In particular, we use $N_0 = 10$, $N_{\max} = 100$, $\beta = 0.0025$, $\eta_0 = 0.05$, $\eta_1 = 0.01$, $r_I = 2$, $r_F = 0.7$ and $t_{\max} = 40000$.

TABLE I
THE ODEN ALGORITHM.

ODEN$(x_I, x_F, N_0, N_{\max}, r_I, r_F)$

1) Initialize the network as:

$$w_0 = x_I;$$
$$w_i = x_I + i \cdot \frac{x_F - x_I}{N_0 - 1}, \quad i = 1, \ldots, N_0 - 1.$$

2) Let $t = 1$, $L_{ins} = 0$, $N = N_0$;
3) Determine $\mathcal{U}_U^t$;
4) If $\mathcal{U}_U^t = \emptyset$, let $t = L_{ins} + \lambda$ and proceed to 11;
5) Randomly choose $w_i \in \mathcal{U}_U^t$;
6) Let

$$r = r_I \cdot \left(\frac{r_F}{r_I}\right)^{t/t_{\max}}; \qquad (1)$$

7) Randomly choose a point $x \in X$ such that $\|x - w_i\| < r$
8) Find $w_j = \arg\min_{w \in \mathcal{U}_U^t} \|x - w\|$;
9) If $F(x) > 0$,

$$w_j = w_j + \eta_0(x - w_j) + \beta(w_{j-1} + w_{j+1} - 2w_j)$$

else

$$w_j = w_j + \alpha \frac{(w_{j+1} + w_{j-1})^{\perp}}{\|w_{j+1} + w_{j-1}\|},$$

where α is a randomly chosen number such that $-\beta \leq \alpha \leq \beta$.
10) If $F(x) > 0$,

$$w_{j\pm1} = w_{j\pm1} + \eta_1(x - w_{j\pm1}).$$

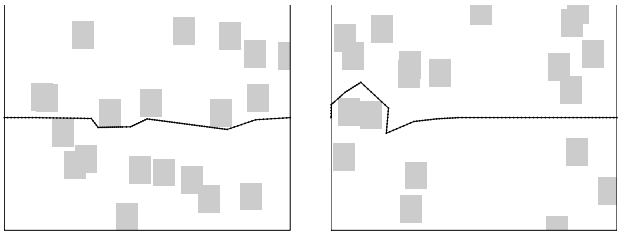
11) If $t - L_{ins} \geq \lambda$,
   a) Let $L_{ins} = t$ and insert a new unit w_{new} such that

$$w_{new} = \frac{w_k + w_{k+1}}{2},$$

   where w_k and w_{k+1} are such that

$$\|w_k - w_{k+1}\| = \max_{w_i \in \mathcal{U}_U^t} \|w_i - w_{i+1}\|.$$

   b) Let $N = N + 1$ and update $\mathcal{U}_U^t$.
12) If $\big(t < t_{\max} \wedge (I \neq \emptyset \vee N < N_{\max})\big)$, let $t = t + 1$ and return to 5, else finish;

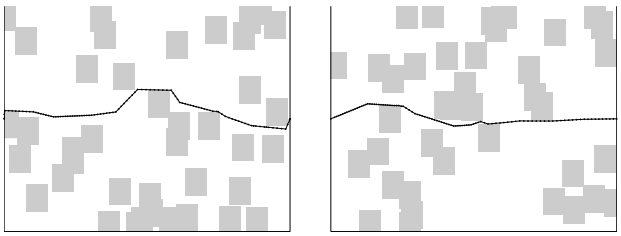


a) b)

Fig. 8. Trajectories obtained with ODEN in environments with 20 random obstacles.

To observe the behavior of the algorithm in environments with little free space, we tested the ODEN algorithm in environments with 60, 80 and 100 random obstacles. Figures 10 and 11 depict the obtained results.

We remark that, as argued in the previous section, the intersection effect depicted in Figure 4 is not observed in the trajectories, even if in many situations the path is close

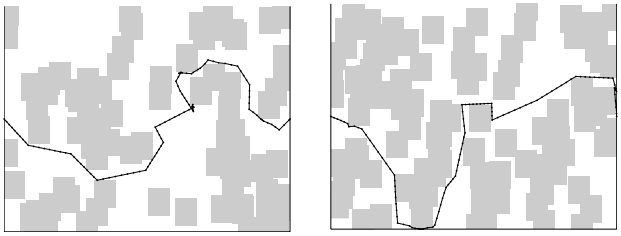


a) b)

Fig. 9. Trajectories obtained with ODEN in two environments with 40 obstacles.

a) b)

Fig. 10. Trajectories obtained with ODEN in two environments with 60 obstacles.

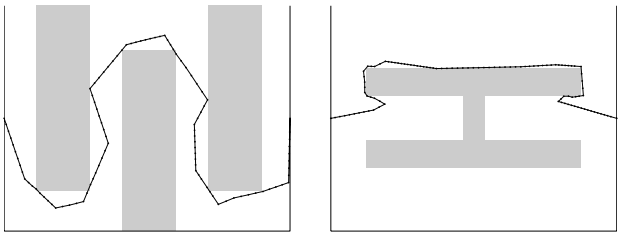


a) 80 obstacles; b) 100 obstacles.

Fig. 11. Trajectories obtained with ODEN in two environments with 80 and 100 obstacles.

to these.

Finally, we present in Figure 12 the results obtained in two environments with non-random obstacles. The distributions of the obstacles in these environments can be seen as possible worst-case situations for the algorithm. Notice that the algorithm is, in fact, able to deliver the expected collision-free paths.



a) b)

Fig. 12. Trajectory obtained with ODEN in environments with non-random obstacles.

In Table III we have reported, for each example, the number of iterations required to reach the final path. We note that the iterations reported in the empty environment arise from the introduction of the additional units necessary to complete the $N_{\max}$ units.

Some remarks are in order. First of all, in the first set of environments, the algorithm is generally able to deliver a solution within few iterations. This is in clear contrast with the original method. Just as a term of comparison, the original algorithm took 2981 and 13954 iterations to produce the two paths depicted in Figure 3. On the other hand, ODEN will generally perform better on more sparse environments than in environments with little free space. This is easily explained if we consider that the algorithm proceeds by sampling the space around the network and pulling the network towards free-space and away from the obstacles. If few sample points lie in free space, more iterations will be required for the algorithm to find a proper path.

On the other hand, "thin" obstacles may be hard to sample and, therefore, to avoid. We produced one further experiment to test the performance of the algorithm when a single thin, long obstacle is found in the environment. The result is produced in Fig. 13. Notice that, even though the environment has a single obstacle and is, in general, a sparse environment, the algorithm took a considerable number of iterations before a solution was found.

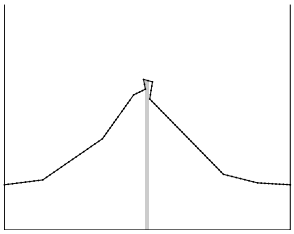


Fig. 13. Trajectory obtained with ODEN in an environment with a thin obstacle.

IV. MODEN: ODEN IN MULTI-ROBOT DOMAINS

We now assume that a set of n robots needs to navigate in a common environment. Given the initial and final positions in configuration space for each of the n robots in the set, we are interested in determining an individual path for each robot, so that the paths are non-intersecting. Intersecting paths may not be a problem, when the dynamics or communication capabilities of the robots allow for collision prevention. For example, it is possible to provide the robots with some signaling mechanism or protocol forcing one of the robots in the intersection to wait for the other.

We focus on the generation of non-intersecting paths with no need for any knowledge of the robot dynamics/communication capabilities. The paths generated by the algorithm can be immediately used by the robot set, without considering any coordination or synchronization mechanism to prevent on-path collisions. Non-intersecting paths may also be desirable by considering once again the problem of the communication channel. If multiple channels are to be set in a common environment, intersecting paths may be undesirable to minimize any interference.

Given the initial and final points in configuration space for each of the n robots, we can apply the ODEN algorithm to each robot individually, yielding n distinct paths, one for each robot. However, these paths will not take into

consideration the existence of the other robots which will probably lead to intersecting paths. The added functionality of MODEN arises from two main ideas: considering the path of other robots as an obstacle and defining an ordering relation to determine the paths for the different robots.

A. Other robots' paths as obstacles

Consider the situation where 2 robots must navigate in a common environment, as depicted in Figure 14. Let R_1 be

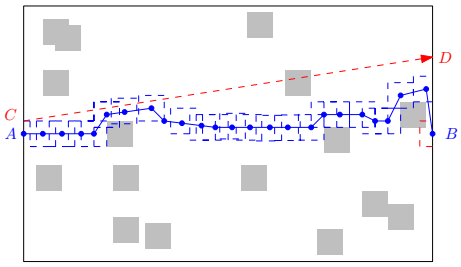


Fig. 14. Situation with 2 robots in a common environment.

the robot traveling from A to B and R_2 be the robot traveling from C to D. If the the trajectory of R_1 is to be avoided when determining the path of R_2, a simple idea it to *consider each PU in the path of R_1 as the center of some obstacle* (see Figure 14). The size of the obstacle can be adjusted as a parameter. This strategy turns the other robot's path into an "obstacle" and we would expect the ODEN algorithm to produce individual trajectories that do not intersect, if possible.

However, this is not exactly so. If ODEN updates all trajectories simultaneously, this will lead to intersecting paths. In fact, consider the situation depicted in Figure 15.a. If the samples used to update each path continue to be

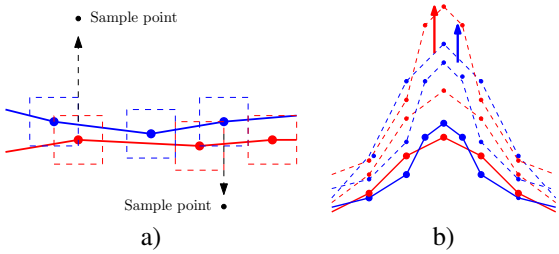


Fig. 15. Situations that may lead to intersection.

sampled in opposite sides of the other path, the final paths will unavoidably intersect. Furthermore, situations may occur in which the paths keep "chasing" each other, as they keep sampling free points in one direction while trying to avoid the other path (see Figure 15.b).

To minimize such phenomenon, we introduce an ordering in the set of the robots and run the ODEN algorithm following that order.

B. Ordering of the robot set

Suppose that a strict ordering among the set of robots is given.[5] According to this ordering, it is possible to refer to each of the robots in the set as $R_1, \ldots, R_n$, where R_i stands for the ith robot in the given ordering. We now successively apply the ODEN algorithm to each of the robots $R_1, \ldots, R_n$ individually: we determine the path for robot R_1 using the ODEN algorithm and considering only the original obstacles in the environment. Then, for each other robot R_i, $i = 2, \ldots, n$, we determine its path using the same ODEN algorithm, but considering as obstacles the trajectories of robots $R_1, \ldots, R_{i-1}$ besides the natural obstacles in the environment.

Finally, to alleviate the intersections arising from cross sampling (see Figure 16), we include a small adaptive bias in the random sampling process.

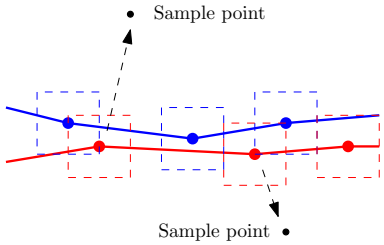


Fig. 16. Intersection will also occur if the depicted sample points are used for update.

C. The MODEN algorithm

Table II presents the MODEN algorithm in pseudo-code. We denote by $\{x_I^1, \ldots, x_I^n\}$ and $\{x_F^1, \ldots, x_F^n\}$ the sets of

TABLE II
THE MODEN ALGORITHM.

MODEN($\{x_I^1, \ldots, x_I^n\}, \{x_F^1, \ldots, x_F^n\}, N_0, N_{\max}, r_I, r_F$)
1) Initialize X_{obs} with the obstacles from the environment;
2) for $i = 1, \ldots, n$ do
3) $P_i = ODEN(x_I^i, x_F^i, N_0, N_{\max}, r_I, r_F)$;
4) Append P_i to X_{obs} as in Figure 14 (the extreme units in the network are not considered);
5) end for

initial and final points for the n robots. P_i is the path generated for robot R_i.

Recall that ODEN uses points randomly sampled from the configuration space to update the units in the network. These points are sampled in a region of radius r around a randomly chosen unit w_i, where r is given by (1). A sample point x is then given by

$$x = w_i + r_x \angle \theta_x,$$

where r_x is a random number between 0 and r, θ_x is a random angle between 0 and 2π and w_i is the chosen unit.

Let $I_1, \ldots, I_k$ be a uniform partition of the interval $I = [0, 2\pi]$. A uniform sampling procedure chooses an angle θ_x in I_i with probability $p_i = 1/k$. Suppose that the chosen angle at some iteration was α_x and the corresponding point

[5]It could be possible to choose this order to optimize some given criteria. In our experiments, either we are given the order or are able to determine it based on the geometry of the initial and final positions of the robots.

x belongs to X_{free}. Then, the probabilities p_i are biased towards the direction α_x according to

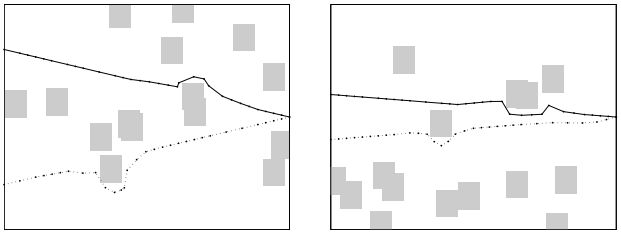
$$p_i = \begin{cases} p_i + \epsilon(1 - p_i) & \text{if } \alpha_x \in I_i; \\ (1 - \epsilon)p_i & \text{otherwise} \end{cases}$$

and then normalized to yield $\sum_i p_i = 1$.

D. Results using the multi-agent algorithm

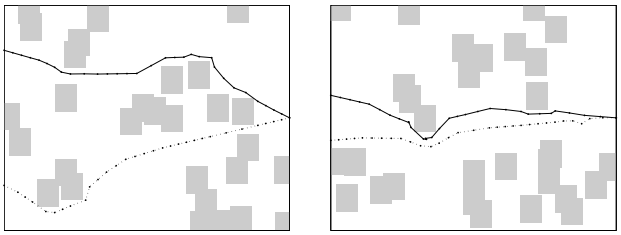
We now present several tests conducted with the multi-robot path planning algorithm in various environments.

Figures 17.a and 17.b present the paths obtained in environments with 15 random obstacles, where the robots depart from different initial positions but arrive to a common final position. Figures 18.a and 18.b present similar results this time obtained in environments with 30 random obstacles.



a) b)

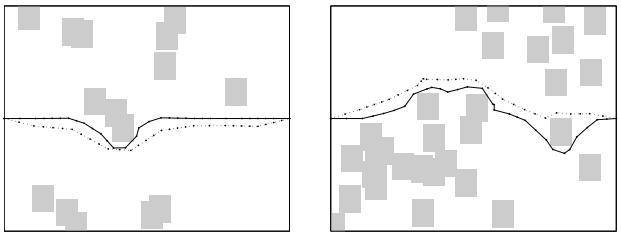
Fig. 17. Trajectories obtained for 2 robots in an environment with 15 obstacles.



a) b)

Fig. 18. Trajectories obtained for 2 robots in an environment with 30 obstacles.

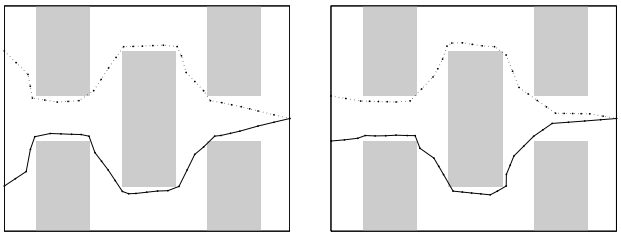
We then applied the algorithm to the extreme situation where the robots depart and arrive at a common location. Notice that this implies that the paths for both robots are coincident in the initial iteration. Figures 19.a and 19.b depict the results obtained in environments with, respectively, 15 and 30 obstacles.



a) 15 obstacles; b) 30 obstacles; obstacles.

Fig. 19. Trajectories obtained for 2 robots with initial coincident trajectory.

Figures 20.a and 20.b depict the results obtained in the 5-obstacle environment of Figure 12.a, for two different sets of initial conditions. We also tested the performance of the algorithm with 3 and 5 robots. Figures 21 and 22 present the obtained results in two different environments.



a) b)

Fig. 20. Trajectories obtained for 2 robots in an environment with non-random obstacles.

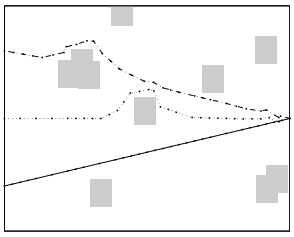
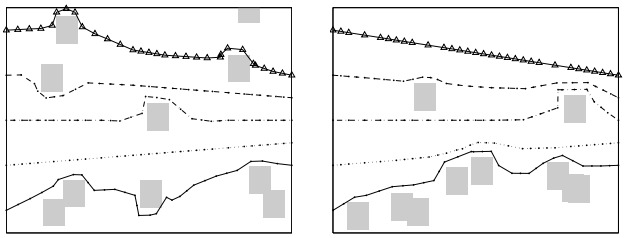


Fig. 21. Trajectories obtained for 3 robots in a random environment.



a) b)

Fig. 22. Trajectories obtained for 5 robots in two environments with 10 random obstacles.

As in Section III, the algorithm was run with the same parameters as those reported in [1]. The dimension of the "virtual obstacles" around each PU is $\frac{1}{3}$ of the size of the actual random obstacles. For each example, the number of iterations to reach the final result is presented in Table III.

V. DISCUSSION

In this paper we have described ODEN, a path-planning algorithm relying on an elastic network and point-based sampling to achieve a collision-free trajectory between two points. The point-samples are used to "pull" the nodes of the network toward free space, and the algorithm is able to achieve a satisfactory trajectory in few iterations.

The main advantages of the ODEN algorithm lie on the simple, intuitive principle in which it relies and its simplicity of implementation. Each unit in the network is *locally* driven away from the obstacles in the environment. ODEN provides

TABLE III

NUMBER OF ITERATIONS FOR THE DIFFERENT EXAMPLES.

Example	Iterations	Example	Iterations
Fig. 7.a	92	Fig. 17.a	168
Fig. 7.b	105	Fig. 17.b	425
Fig. 8.a	215	Fig. 18.a	470
Fig. 8.b	264	Fig. 18.b	652
Fig. 9.a	291	Fig. 19.a	249
Fig. 9.b	155	Fig. 19.b	920
Fig. 10.a	1285	Fig. 20.a	1609
Fig. 10.b	1590	Fig. 20.b	1073
Fig. 11.a	3608	Fig. 21.a	517
Fig. 11.b	3792	Fig. 22.a	1259
Fig. 12.a	1891	Fig. 22.b	928
Fig. 12.b	2004		
Fig. 13	721		

simple solutions with no *a priori* knowledge of the geometry of the environment, unlike other path planning algorithms.

From the obtained results, we claim ODEN to be an efficient algorithm, being able to provide collision-free paths even in environments with a large percentage of occupied area. In a practical application, ODEN would simply require a sampling function to evaluate whether a point in configuration space is in free-space or inside an obstacle. This information can easily be retrieved from sensorial data and makes this method simple to implement.

However, and as argued in Section II, the sampling mechanism supporting ODEN implies that the algorithm is less effective in environments with little free-space or with very thin obstacles. In the former case, it will be hard to sample points in free-space to pull the network away from the obstacles. In the latter case, it will be hard to sample the obstacles so as to drive the network away from them.

MODEN extends the simple principle behind ODEN to multi-agent scenarios, while remaining a local method driven by obstacles. We have considered multi-robot path-planning problems in which the main goal is to avoid *path intersection*. This is not such a usual approach, as more elaborate ways exist to coordinate multiple robots wandering in a common environment (see, for example, the approaches in [15]–[17]). However, this approach adequately verifies the applicability of ODEN to more complex problems. The aforementioned approach suitably illustrates the simplicity and local behavior of MODEN and provides a better grasp on the effectiveness of ODEN's underlying working principle. Nevertheless, and in spite of the encouraging results, we should remark that if the initial straight-line path of the several robots intersects, it is generally not possible to ensure a non-intersecting solution. It may also happen that the path for the initial robot renders the task of finding a collision-free infeasible.

In the final version of the paper, we will compare the performance of ODEN and MODEN with that of other path-planning methods. This comparison with other well-established path-planning methods will allow us to further understand the applicability of these algorithms.

It would also be of interest to explore the use of richer attraction functions F. If the attraction function F is more than an "obstacle-indicator", the algorithm may use the extra information to drive the updates in a more informed fashion and thus improve its performance.

ACKNOWLEDGEMENTS

This work was partially supported by the POS_C financing program, that includes FEDER funds. The first author acknowledges the PhD grant SFRH/BD/3074/2000. Manuela Veloso is currently in sabbatical leave at Radcliffe Institute for Advance Study (Harvard University) as a Sargent-Faull Fellow.

REFERENCES

[1] J. Moreno and M. Castro, "Heuristic Algorithm for Robot Path Planning Based on a Growing Elastic Net," in *Proc. 12th Portuguese Conf. Artificial Intelligence, EPIA 2005*, 2005, pp. 447–454.

[2] J. Barraquand, L. Kavraki, J. C. Latombe, T. Li, R. Motwani, and P. Raghavan, "A Random Sampling Scheme for Path Planning," *Int. J. Robotics Research*, vol. 16, no. 6, pp. 759–774, 1997.

[3] Y. Kanayama and B. Hartman, *Smooth Local Path Planning for Autonomous Vehicles*, 1990, pp. 62–67.

[4] G. Walsh, R. Montgomery, and S. Sastry, "Optimal Path Planning on Matrix Lie Groups," in *Proc. 33rd IEEE Conf. on Decision and Control*, vol. 2, 1994, pp. 1258–1263.

[5] S. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," Computer Science Department, Iowa State University, Tech. Rep. TR 98-11, 1998.

[6] J. Bruce and M. Veloso, "Real-Time Randomized Path Planning for Robot Navigation," in *Proc. 2002 IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2002, pp. 2383–2388.

[7] L. Kavraki, M. Kolountzakis, and J. C. Latombe, "Analysis of Probabilistic Roadmaps for Path Planning," in *Proc. 1996 IEEE Int. Conf. Robotics and Automation (ICRA'96)*, 1996, pp. 3020–3025.

[8] J. Barraquand and J. C. Latombe, "A Monte-Carlo Algorithm for Path Planning With Many Degrees of Freedom," in *Proc. 1990 IEEE Int. Conf. Robotics and Automation*, 1990, pp. 1712–1717.

[9] S. Caselli, M. Reggiani, and R. Rocchi, "Heuristic Methods for Randomized Path Planning in Potential Fields," in *Proc. 2001 IEEE Int. Symp. Computational Intelligence in Robotics and Automation*, 2001, pp. 426–431.

[10] Y. Hwang and N. Ahuja, "A Potential Field Approach to Path Planning," *IEEE Trans. Robotics and Automation*, vol. 8, no. 1, pp. 23–32, 1992.

[11] A. Yahja, S. Singh, and A. Stentz, "Recent Results in Path Planning for Mobile Robots Operating in Vast Outdoor Environments," in *Proc. 1998 Symp. Image, Speech, Signal Processing and Robotics*, 1998.

[12] A. Stentz, "Optimal and Efficient Path Planning for Partially-Known Environments," in *Proc. 1994 IEEE Int. Conf. Robotics and Automation*, vol. 4, May 1994, pp. 3310–3317.

[13] S. Kambhampati and L. Davis, "Multiresolution Path Planning for Mobile Robots," *IEEE J. Robotics and Automation*, vol. RA-2, no. 3, pp. 135–145, 1986.

[14] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, *Principles of Robot Motion*. MIT Press, 2005.

[15] M. Bennewitz, W. Burgard, and S. Thrun, "Optimizing Schedules for Prioritized Path Planning of Multi-Robot Systems," in *Proc. 2001 IEEE Int. Conf. on Robotics and Automation*, vol. 1, 2001, pp. 27–276.

[16] S. LaValle and S. Hutchinson, "Optimal motion planning for multiple robots having independent goals," in *Proc. 1996 IEEE Int. Conf. Robotics and Automation*, 1996, pp. 2847–2852.

[17] P. Švestka and M. Overmars, "Coordinated path planning for multiple robots," *Robotics and Autonomous Systems*, vol. 23, pp. 125–152, 1998.