Proceedings of the 2006 IEEE/RSJ
International Conference on Intelligent Robots and Systems
October 9 - 15, 2006, Beijing, China

Spline-Based Robot Navigation

Evgeni Magid
Applied Mathematics
Technion - Israel Institute of Technology
Haifa, Israel
Email: evgenue@tx.technion.ac.il

Daniel Keren
Computer Science
University of Haifa
Haifa, Israel
Email: dkeren@cs.haifa.ac.il

Ehud Rivlin and Irad Yavneh
Computer Science
Technion - Israel Institute of Technology
Haifa, Israel
Email: {ehudr, irad}@cs.technion.ac.il

***Abstract*— This paper offers a path planning algorithm based on splines. The sought path avoids the obstacles, and is smooth and short. Smoothing is used as an integral part of the algorithm, and not only as a final improvement to a path found by other methods. In order to avoid a very difficult optimization over all the path's points, it is modeled by a sequence of splines defined by a gradually increasing number of knots.**

I. Introduction

Motion planning is concerned with automatic planning of a collision-free path between initial and final configurations. The classical motion planning problem, termed the *piano movers problem*, is defined for complete *a priori* information about the obstacles in the environment. The piano movers model is formulated as follows [15]. Given are a solid object of known size and shape in two- or three-dimensional space, its initial and target position and orientation, and a set of obstacles in the environment. The shapes, positions and orientations of the obstacles in space are fully described. The task is to find a continuous path for the object from its initial position to the target position, while avoiding collisions with obstacles along the way. Because full information is assumed, the whole operation of path planning is a one-time off-line operation. The basic requirements become *soundness* (collision free path), *completeness* (guaranteed to find a path if it exists), *optimality* (being close to the optimal path) and *complexity* (time and space performance). The main difficulty of the piano movers model is to obtain a computationally efficient scheme.

Searching is a fundamental component of piano movers conception. Given a search space, a set of possible problem states, and a state transition function to determine the states directly reachable from any given state, a search method is an algorithm to control the exploration of the state space in order to identify a path from an initial state to the goal. Given techniques to search a state space for a path, it remains to take an environment and to construct a state space to represent it [6]. A straightforward approach is to take a geometric representation of a free space and to discretize it (e.g., [2]). Other mechanisms of mapping the robot's environment onto a discrete searchable space include visibility graph (e.g., [14]) and Voronoi diagram (e.g., [5]) construction techniques. Rather than searching through a discrete space that represent the state of the robot, an alternative is to model the configuration space of the robot as a continuous space. Path planning is considered as the appropriate trajectory within this continuum, modeled, for example, as a potential field (e.g. [7], [9], [14]).

An apparent advantage of the piano movers approach is that any optimization criteria can be easily introduced: finding the shortest path, or the minimum-time path, or the safest path, or smoothest path etc. Smoothness of the path is essential for mobile robot navigation, because non-smooth motions can cause slippage of wheels which degrades the robot's dead-reckoning ability. Given a smooth path, a robot can move for a long distance without receiving extra visual or range information. The smoothness property of the path is extremely important for car-like vehicles, which are constrained with their motion abilities.

Numerous motion planners consider the car-like vehicle as a three-dimensional system moving in the plane and subjected to constraints on the curvature in addition to the non-holonomic constraint of rolling without slipping. The pioneering work by Dubins [3] showed that the minimal length paths for a car-like vehicle consist of a finite sequence of two elementary components: arcs of circle and straight line segments (e.g., [19]). From then, almost all of the proposed motion planners compute collision-free paths constituted by such sequences [13]. As a result, the paths are piecewise C^2: they are C^2 along elementary components, but the curvature is discontinuous between two elementary components. To follow such paths, a real system has to stop at these discontinuity points in order to ensure the continuity of the linear and angular velocities. One of the solutions is to smooth the sequences straight line-arc of circle by clothoids (e.g., [4]). The paths are then C^2 between two cusp points. Unfortunately, clothoids do not have a closed form making the control of their shapes difficult and dangerous in the presence of obstacles. A completely different approach is to obtain a shortest path with a visibility graph methods (e.g., [10]) or Generalized Voronoi graph (e.g., [17]), with further modification of the path according to dynamic constraints (e.g., [1]).

Our research goal was to create a path-generating method for a car-like mobile robot. It should reach a target configuration from a certain initial configuration as fast as possible under known environment without collision. The main objective of the new method is the smoothness of the path, while the optimality (i.e. path length) is only secondary. We introduce the smoothness requirement on a path from the first stage of the algorithm instead of smoothing the path on the last

1-4244-0259-X/06/$20.00 ©2006 IEEE

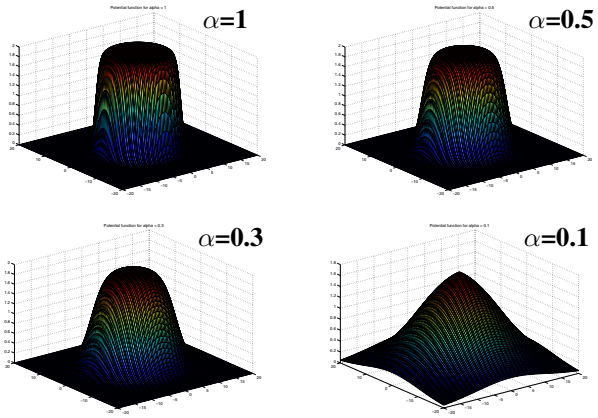

Fig. 1. The repulsive potential function (eq.(1)) of a single obstacle with a center $(x, y) = (0, 0)$ and radius $\rho = 10$ as a function of robot's coordinates $(x(t), y(t))$. The parameter α influences the slope of the potential field.

stages only. A collision free but not sufficiently smooth path is considered to have the same low quality as a path which intersects obstacles of the environment.

II. The Cost function

The algorithm navigates a point robot in a planar known environment populated by stationary obstacles. Our goal is to find a path from the starting point S to a target point T. The environment consists of circles of different sizes which may intersect each other. We believe that most real environment can be approximated with such simplified model. In sec.VI we will deal with more complicated obstacles.

To ensure the collision free path for each obstacle a repulsive potential function with a high value inside the obstacle and on its boundary and a small value on the free space is defined. A potential field with high potential in the obstacles' center "pushes" the path outside. The boundary of the obstacle is a breaking point of the potential function. At the boundary potential field begins to decrease drastically outside the obstacle with distance and becomes zero fast enough in a close vicinity of the obstacle. The following function defines the repulsive potential function of a single obstacle at the robot's configuration $q(t) = (x(t), y(t))$ and satisfies all the requirements:

$$U_{rep}(q) = \frac{\beta}{2}(1+\tanh(\alpha(\rho-\sqrt{(x(t)-x)^2+(y(t)-y)^2}))), \quad (1)$$

where ρ is the radius of the obstacle with the center at (x, y). The additive constant in the brackets makes the potential function non-negative. A scaling factor β shows the influence of the obstacle penalty on the path cost. Division by 2 normalizes the expression in the brackets.

A parameter α is responsible for pushing the path outside the obstacle. With $\alpha = 1$, close to the boundary - both inside and outside the obstacle - the potential function changes fast. While the potential field rapidly converges to zero outside the obstacle as a point moves away from the obstacle's center, the

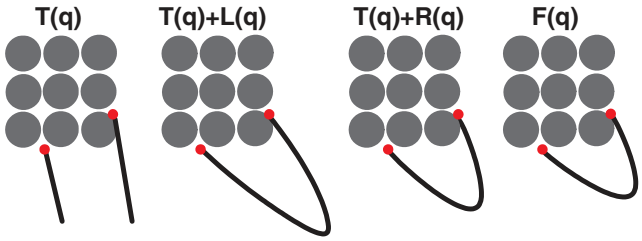

Fig. 2. The influence of different components of path cost function on the solution.

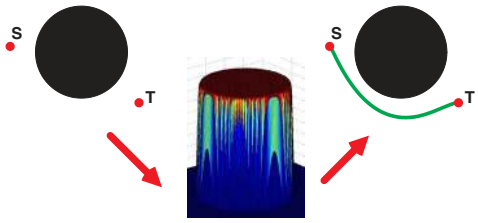

Fig. 3. Start, target and environment together with the potential function define the final path for a mobile robot.

situation is problematic inside the obstacle. Since close enough to the center of the obstacle the potential field is uniform over a large region, there is no "pushing out" of the path from the center of the obstacle toward its boundary. Thus, to ensure different kinds of "pushing out", we need adjust the parameter α while searching for the path. For a small value of α the function changes slowly; when α is large, the peaks of the potential function are extremely strong within the obstacle(Fig.1).

Topology $T(q)$ is the function that takes into an account all obstacles of the environment. The obstacles that are far enough from the point of the path $(x(t), y(t))$ do not influence the path cost, since the values of such expressions U^j_{rep} are very small. To obtain all points of the path, we integrate equation(1) by parameter t, taking into account the length of each segment, and sum over all N obstacles of the environment:

$$T(q) = \sum_{j=1}^{N} \int_{t=0}^{1} U^j_{rep}(q) \cdot \sqrt{(x'(t))^2 + (y'(t))^2} dt, \quad (2)$$

Roughness $R(q)$ is responsible for the smoothness property of the path and penalizes in the case of a non smooth path. The square root provides a regularization of the standard smoothness measure with respect to other measures:

$$R(q) = \sqrt{\int_{t=0}^{1} ((x''(t))^2 + (y''(t))^2) dt}, \quad (3)$$

The last term is a path length $L(q)$:

$$L(q) = \int_{t=0}^{1} \sqrt{(x'(t))^2 + (y'(t))^2} dt, \quad (4)$$

The final function of the path cost[1] is:

$$F(q) = T(q) + R(q) + 0.5 \cdot L(q), \tag{5}$$

Roughness and *path length* measures are used to satisfy our requirements on the quality of the path. Fig. 2 shows the importance of each of the three components. When only T(q) is used, the path may become extremely long, since the only requirement on the path is to stay far from obstacles. T(q) together with L(q) generate a path which is enforced to be shorter than the latter. T(q) together with R(q) is close to the path cost function containing all three components and the difference is very subtle in simple cases. In more complicated examples the influence of the additional length term accumulates through the iterations and the resulting paths may become completely different. The lack of L(q) may lead into a long self-crossing sections of the path which lie far from all obstacles (i.e. T(q)→0) and all efforts of the algorithm are concentrated on minimization of R(q) component. These three desired properties of the path, implemented with a variational planning approach, are built into our potential cost function which guides the robot through the environment(Fig.3).

III. The Algorithm

The algorithm works iteratively, starting from two given points of the path; start point S, target point T and the obstacles of the environment serve as input for the algorithm. At the first step the initial guess on the path is made. The initial path is evaluated. This evaluation serves as a "minimum path cost available at the moment" for further improvement. At each iteration a new initial guess based spline is proposed as a better option for the path. Optimizing the initial proposal with Nelder-Mead Simplex Method(NMSM) [12] to minimize a cost of the path results in a better path, which in turn serves as an initial guess for the next iteration. The optimization deals only with the via points of the path, which define the spline, while the evaluation of the path takes into an account all points of the path. Each iteration rebuilds the spline, utilizing the information from the previous stage, increasing the number of spline's points and adjusting parameters of the minimization target function. After each iteration the path quality is tested relative to the previous iteration. As soon as the path satisfies the basic property of being collision free, a few more iterations are conducted in order to improve the resulting path locally.When the convergence of the path cost function stops, the algorithm informs that the path is found. If a collision free path is not obtained after a predefined number of iterations, the algorithm terminates (Fig.4).

Via points (MP) are defined as evenly spaced points along the curve (S,T). First via point MP divides the initial path (S,T) in two equal parts. Two via points create 3 equal parts

[1]Our choice of constants is driven by our wish to pay most attention to the main requirement of the collision free path and smoothness of the path. The length is less important, since, while the search proceeds in the vicinity of the obstacles, minimization of the path length is an inherent property of *topology* component. As soon as *topology* component is close to zero, $0.5 \cdot L(q)$ still prevents the infinite growth of the path length in a fully satisfactory way.

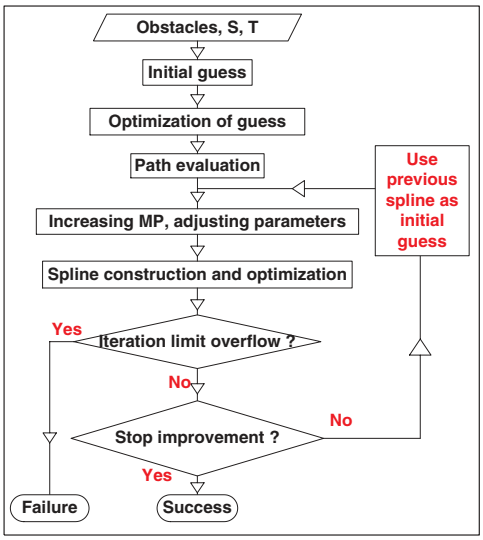

Fig. 4. Algorithm flow block.

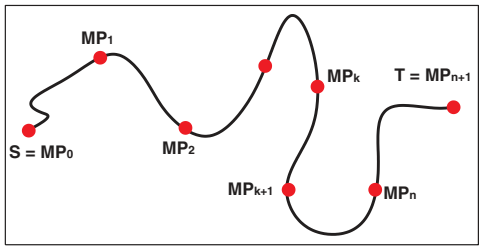

Fig. 5. The distribution of the via points MP_i for $i = 0..n+1$ is uniform.

of the path and so on. This way when we acquire n via points, the path (S, T) is divided into $n+1$ equal segments. Eventually we posses a uniform distribution of *via points* on the path length. (Fig.5)

The first step of our algorithm is an initial guess of the path. We simply choose it to be a straight line segment (S,T). Then the NMSM attempts to minimize the cost function of n real variables (via points) using only function values, without any derivative information. It can often handle discontinuity, particularly if it does not occur near the solution, but it may only give local solutions.

Iterations are the heart of our algorithm. The main idea is that each new iteration is getting closer and closer to the desired path. We start from a small value of parameter β, which shows how much the obstacle should be considered while choosing the path. Further β is increased: the importance of collision with the obstacles grows with each iteration. Similarly, we start from a small value of α, increasing it with time. The third varying parameter of an iteration is the number of NMSM iterations (NMSM-iterations). The last and probably the most important parameter which is increased at each new iteration is the number of path *via points* MP, which determines the complexity of the spline. At each iteration we use a result of the previous iteration as an initial guess for NMSM, implemented in MATLAB's function $fminsearch$.

At the first iteration of stage one $via\ point$ is chosen. It serves as an initial guess to the optimization function. The target function of the optimization is a cost of the spline, based on the start, target and via points. The NMSM optimization "plays" with the via points, moving them locally in different directions in order to minimize the cost of the produced spline. When the number of optimization attempts prevails over the number of NMSM-iterations allocated for this iteration of the algorithm, a spline with a minimal cost path (relatively to other attempts) is chosen as a next initial guess. The $via\ points$ are uniformly redistributed on the length of this spline (fig.5) and serve for the next spline optimization. At each iteration the number of via points is increased by one, enriching the spline. The spline becomes more flexible with regard to its ancestor. Parameters α and β are increased as well. In the several first iterations when parameter β still has a low value, the main terms of the cost function are $roughness$ and $path\ length$. As parameter β grows, the main term of the path cost becomes intersection with obstacles. Each obstacle adds its own penalty to the cost function. In case of intersecting obstacles the cost of passing through the intersection is a sum of each separate intersecting obstacle's contribution.

The number of NMSM-iterations varies with each iteration as follows. Nelder-Mead algorithm tends to work well in practice by producing a rapid initial decrease in function values, but it does not ultimately converges to a minimizer [12]. This means that first NMSM-iterations are significant, while the following NMSM-iterations decrease the cost function less essentially. During the experiments we noted that in the first 3-4 iterations the most significant part of the work is accomplished, when the spline is rather simple. Since the number of via points is small in the first iterations, the computational cost of every NMSM-iteration is not high. This allows us to allocate more NMSM-iterations in the first iterations of the algorithm, which create the "skeleton" of the final path. Further iterations have more local variations around this "skeleton", coiling round single obstacles one by one. The fluctuations of the path (i.e. the improvement of the path cost) are less essential, while the computation cost of each NMSM-iteration increases rapidly with the number of via points. Thus, less NMSM-iterations are wholly satisfactory in the further iterations.

The iterations may stop in two cases: success and failure. The algorithm decides that it succeeded to find a path when after a number of iterations the collision free path is obtained. There are two stages of the algorithm, which are independent of each other. Both stages have the same structure and the main difference is a particular choice of parameters α and β. If the first stage, supposed to deal with simple environments, fails, the second stage, supposed to deal with more sophisticated environments, restarts all the process from the beginning in attempt to find a path far away from the obstacles. As soon as a collision free path is obtained, further iterations are used to improve the obtained path locally in the terms of its length and smoothness. This reminds of a standard procedure used in methods for searching a shortest smooth path - smoothing a final path. However, in our algorithm the requirement of smoothness is introduced from the beginning of the search and not only on the final stages.

The evaluation function, based on eq.(5), detects when the so called *false improvement* of the path starts:

$$EF(q) = R(q) + L(q), \tag{6}$$

The distance from the obstacles is not included at all and only the quality of the produced path is considered. As soon as the next iteration, improving the path from the obstacle avoidance point of view, increases the value of eq.(6), the evaluation function signals that it is time to stop the iterations.

IV. Local minima

In some cases our algorithm fails to find an existing path from S to T in the given environment in the first stage due to a local minimum of the cost function. The optimization of path cost function is conducted over a space of much larger dimension (the number of spline via points) and is quite costly. Even though Nelder-Mead optimization uses only function values without any derivative information, it often produces a local solution in concave scenes in the presence of local minimum of a potential field, which is not globally optimal. Even though for general non-convex functions the Nelder-Mead algorithm tends to work well in practice by producing a rapid initial decrease in function values, it is an open problem if there exists any function $f(x)$ in R^2 for which the algorithm always converges to a minimizer [12].

To globalize the Nelder-Mead method, probabilistic restarts, utilizing a memory of previous iterations, are introduced in [18]. Unfortunately, this solution has several serious drawbacks, which are common to all randomized planning methods [14]. The planner typically generates different paths if it is run several times with the same problem and the running time varies from one run to another. If the input path planning problem admits no solution, the planner has no way to recognize it even after a large amount of computation. Hence, a limit on the running time of the algorithm has to be imposed. But, if the limit is attained and no path has been generated yet, there is no guarantee that the free path does not exist.

For the latter reason we did not apply a randomized method for restarting the application with different initial guesses - it is hard to establish a good trade off between infinite number of restarts when the free path exists and the situation with no free path. Any such choice would fail in some generic case. Our solution is simple and successfully deals with many cases of potential function local minima. When the first stage of the algorithm signals its failure, we already know that it is caused by a local minima of the potential field. It means that the potential field map requires a reconstruction. We take different scale of α and β variables to rearrange local minima of the potential field. With small values of α potential field becomes significant in all populated regions of the environment, smoothing the local variations of the field. The intersecting obstacles create not a single and strong local

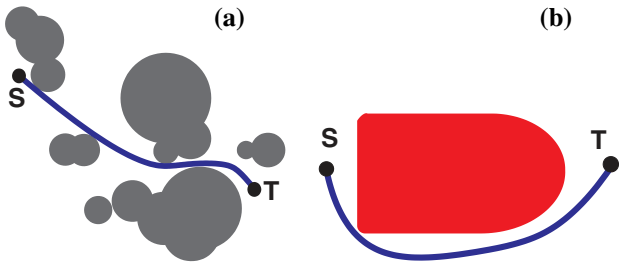

Fig. 6. Navigation examples: (a) - with eq.(1), (b) - with eq.(10)

spike of the field, but a less distinguishing and larger region. Now every populated region looks on a potential field map as a big obstacle with small and rather local variations. New local minima of the field may be obtained between the populated regions, which definitely correspond to the free space of the environment. Yet this can not solve the problem when the start and target points are situated deep inside the populated regions. In such cases the only possible solution is an infinite sequence of initial guess restarts, which we explained and rejected earlier.

V. Experimental results

The simulation supports 2 types of maps: the maps consisting of a finite number of circles (an existing map or online created map) and binary image maps (the details are presented in the next section). Maps of the first type consist of convex, simple concave (intersecting-by-pairs only circles) and complicated concave environments.

In practice,we start with parameters $\alpha = 0.5$ and $\beta = 4$ for the first iteration of the first stage, consisting of 14 iterations, and increase them further. Parameter α takes values 0.6, 0.7 and 0.8, while β is doubled on each iteration before the switch to a next α and then restarts from 32. The number of optimization iterations starts from 250 with a further decrease to 100. The second stage, consisting of 36 iterations, starts with parameters $\alpha = 0.05$ and $\beta = 4$ for the first iteration with a further increase. Parameter α takes values 0.1, 0.3, 0.5, 0.6, 0.7 and 0.8, while β is doubled at each iteration before the switch to a next α and then restarts from 4. The number of optimization iterations starts from 250 with a further decrease to 100 (for details refer to [16]).

Our experiments in convex and simple concave environments showed fast convergence of the method in all scenes within the first stage of the algorithm (fig.6(a)). Tests in complicated concave environments showed that the problem of getting stuck in the local minimum is solved by our algorithm in the second stage for the majority of (S,T)-points choices. In the course of the simulations, we identified and partially solved the following significant problems, inherent to all Potential Field Methods [11] and independent of the particular implementation:

Trap situations due to local minima occur when the robot runs into a dead end. Traps can be created by a variety of different obstacle configurations, and different types of traps can be distinguished. Simple trap-situations are resolved with our map reconstruction method on the second stage of the algorithm. More complicated trap-situations can be resolved by heuristic or global recovery.

No passage between closely spaced obstacles. A mobile robot attempts to pass between two closely spaced obstacles. The repulsive fields $U^1_{rep}(q)$ and $U^2_{rep}(q)$ are combined and the sum of them in the opening appears to exceed the penalty of each separate obstacle. This problem fully depends on the selection of coefficient α. It does not arise in the first stage of our algorithm (simple cases) when α is relatively large. In the second stage it appears in the beginning and disappears further with the growth of α.

VI. New Potential Function

The solution proposed in sec.II assumes that each obstacle is a circle or a union of a finite number of intersecting circles. This insures a simple definition of a repulsive potential as a function of distance from the center of the circle. Unfortunately, this simplicity plays a significant role in local minimum trapping: each pair of intersecting circles creates a local minimum of the potential field. The more obstacles intersect, the more local minimum of the field are created.

A new repulsive potential function is not constrained to a specific form of the obstacles any more. Given a set A, the potential function $f(A)$, based on [8], is guaranteed to obtain small values on A and to increase as we move away from A. Thus, $e^{-f(A)}$ is exactly the potential function we need.

Given a binary image map, the algorithm works in two stages. At the first stage a function $g(x, y)$, responsible for the potential field map of the environment, is created.The binary image map of a size $n \times m$ is transferred into a discrete set of the points of the environment, with a pixel-by-pixel sampling. A set S of points s_i is obtained; s_i refers to a point of an obstacle or its boundary in the map:

$$S = \{s_i\} = \{(x_i, y_i)\}, i = 1..N, 0 \leq N \leq nm \qquad (7)$$

Since the influence of the map size is very extreme, map normalization is necessary. After sampling, the center of obstacles' weight is placed into the center of a square R with a side of 4 units. The image map is normalized to fit into R. Then R is provided with a coordinate system with the origin in its center; now it contains all obstacles of the environment and a part of a free space and serves as a domain of integration in eq.(8). On the set S we define a collection $P_h(x,y)$ of all polynomials of power h in x and y. Experimentally we found out that $h = 5$ is fully satisfactory: smaller h gives a coarse approximation, while for larger h the influence of lower powers components is negligibly small with regard to higher powers components. $P_h(x,y)$ is used to construct a positively defined matrix **A** of size 21×21, where each element $A_{l,m}$, of A is computed as follows for all $1 \leq l, m \leq 21$:

$$A_{l,m} = \sum_{i=0}^{N} f_l(s_i) f_m(s_i) + \lambda \iint_R \left(\frac{\partial f_l(s)}{\partial x} \frac{\partial f_m(s)}{\partial x} + \frac{\partial f_l(s)}{\partial y} \frac{\partial f_m(s)}{\partial y} \right) dxdy \quad (8)$$

The parameter λ, obtained experimentally (λ=0.001), is a positive constant, responsible for the stable solution. When λ has a high value, the potential field does not correspond tightly to the obstacle. Decreasing λ makes a better correspondence. As soon as λ=0 the solution is not stable anymore. A function $g(x, y)$ is obtained as follows:

$$g(x, y) = (f_1, f_2, ..., f_k) \cdot A^{-1} \cdot (f_1, f_2, ..., f_k)^T; \quad (9)$$

Function $g(x, y)$ obtains small positive values within the obstacles and high positive values in the free space. The potential function at the robot location $q = (x, y)$ is defined as follows:

$$U_{rep}(q) = e^{-g(x,y)k\beta} \quad (10)$$

with further integration on the path length for all configurations. Coefficient β decreases in two times at each iteration of the algorithm, starting with β=32; coefficient $k = 100$ was obtained experimentally and remains constant.

In the second stage the algorithm presented in sec.III is applied for a search; eq.(10) replaces eq.(1) and no further summing in eq.(5) is needed, since $g(x, y)$ takes into an account all obstacles of the environment by definition. The two stages are executed separately, since the first stage demands exhaustive calculations which could be avoided in the case the same map is used again. We successfully tested the navigation abilities of our algorithm with the new potential function in several environments (Fig.6(b)).

VII. Conclusion

In this paper we present a new path-generating method for a car-like mobile robot under the piano movers model. The robot reaches the target from a certain initial configuration in a known environment without obstacles collision. The algorithm is based on a variational planning method. Local minima problem is partially solved by rebuilding the potential field map. The obtained path was almost as short as the optimal path and much smoother than the latter, which would guarantee that the target can be reached by a car-like vehicle.

Finally, we present a new repulsive potential function which is not constrained to a specific form of the obstacle. The function is applied directly without any initial approximation of the obstacle which allows to use almost any binary image as a navigational map. We successfully tested the navigation abilities of our algorithm with the new potential function in several simple environments.

References

[1] S. Aydin and H. Temeltas. A novel approach to smooth trajectory planning of a mobile robot. IEEE AMC, 472-477, 2002.

[2] M. Berg, M. Kreveld, M. Overmars and O. Schwarzkopf. Computational Geometry: Algorithms and Applications. Springer-Verlag, 2000.

[3] L. E. Dubins. On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents. Amer. J. Math.,79:497-516, 1957.

[4] S. Fleury, P. Soueres, J.-P. Laumond and R. Chatila, Primitives for smoothing mobile robot trajectories. IEEE Trans. of Robotics and Automation, 11:441-448, 1995.

[5] S. Fortune, A Sweeping Algorithm for Voronoi diagrams. ACM Symposium on computational geometry, 313-322, 1986.

[6] G. Dudek and M. Jenkin. Computational principles of mobile robotics. Cambridge University Press, 2000.

[7] Y. K. Hwang and N. Ahuja. Path Planning Using a Potential Field Representation. IEEE ICRA, 648-649, 1988.

[8] D. Keren and M. Werman. Probabilistic Analysis of Regularization. IEEE PAMI, 15(10):982-995, 1993.

[9] O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. International Journal of Robotics Research, 5(1):90-98, 1986.

[10] T. Kito, J. Ota, R. Katsuki, T. Mizuta, T. Arai, T. Ueyama and T. Nishiyama. Smooth path planning by using visibility graph-like method. IEEE ICRA, 3770-3775, 2003.

[11] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for Mobile Robot Navigation. IEEE ICRA, 1398-1404, 1991.

[12] J. C. Lagarias, J. A. Reeds, M. H. Wright and P. E. Wright. Convergence Properties of the Nelder-Mead Simplex Method in Low Dimensions. SIAM Journal of Optimization, 9(1):112-147, 1998.

[13] F. Lamiraux and J.-P. Laumond. Smooth Motion Planning for Car-Like Vehicles. IEEE Trans. of Robotics and Automation, 17(4):498-502, 2001.

[14] J. C. Latombe. Robot Motion Planning. Kluwer Academic Publishers, USA, 1991.

[15] V. J. Lumelsky and A. A. Stepanov. Path Planning Strategies for a Point Mobile Automaton Moving Amidst Unknown Obstacles of Arbitrary Shape. Algorithmica, 2:403-430, 1987.

[16] E. Magid, D. Keren and E. Rivlin. Autonomous Robot Navigation. Master thesis, Technion, Israel, 2006.

[17] K. Nagatani, Y. Iwai and Y. Tanaka. Sensor based navigation for car-like mobile robots using Generalized Voronoi Graph. IEEE/RSJ IROS, 1017-1022, 2001.

[18] S. Wolff. A local and global, constrained and simple bounded Nelder-Mead Method. Tech.report,Weimar University, Germany, 2004

[19] A. Zelinsky and I. Dowson. Continuous smooth path execution for an autonomous guided vehicle (AGV). IEEE Region 10 Conference, Tencon 92, 871-875, 1992.