# Simulating Pedestrian Behavior with Potential Fields

Fábio Dapper[1], Edson Prestes[1], Marco A.P. Idiart[2], and Luciana P. Nedel[1]

[1] Instituto de Informática, Universidade Federal do Rio Grande do Sul,
[2] Instituto de Física, Universidade Federal do Rio Grande do Sul,
Porto Alegre – RS – Brazil
{fdapper,prestes,nedel}@inf.ufrgs.br, idiart@if.ufrgs.br

**Abstract.** The main challenges of realistically simulating the displacement of humanoid pedestrians are twofold: they need to behave realistically and they should accomplish their tasks. Here we present a field potential formalism, based upon boundary value problems, that allows a group of synthetic actors to move negotiating space, avoiding collisions, attaining goals in prescribed sequences while at same time producing very individual paths. The individuality of each pedestrian can be set by changing its inner field parameters. This leads to a broad range of possible behaviors without jeopardizing its task performance. Simulate situations as behavior in corridors, collision avoidance and competition for a goal are presented and discussed.

## 1 Introduction

The use of synthetic actors able of acting as autonomous agents in applications involving virtual environments is becoming more and more common [1]. Suitable skills for those actors (often simulating human beings) include: a realistic appearance, the ability to produce natural movements, and the aptitude to reasoning and act in an unforeseeable way.

To simulate the behavior of human beings, it is usual to consider system architectures implemented in layers. The lowest one deals with the rotation of each body joint. The intermediary level is responsible for encapsulating composed movements that bring together a set of single joint motions. These movements can represent simple tasks (e.g. stand-up, sit-down, take something, give a step) that used together can provide a higher abstraction level, so called behaviors (e.g. open the door, walk from one position to another one, etc). Finally, the higher abstraction motion layer (cognitive) involves a reasoning mechanism that makes decisions and commands actions in view of the context information (e.g. position, orientation, and distance to target) and humanoids intentions, beliefs and desires.

In a previous work [2] we presented a well succeeded proposal for the implementation of the cognitive level using the BDI (beliefs, desires and intentions) architecture to simulate autonomous agents reasoning. However, good solutions for lower level behaviors may also be investigated. Such solutions should preview

not just a handy manner to specify complex tasks based on simple ones, but also to consider the addition of expressiveness on those tasks.

The simulation of virtual humans moving into a synthetic world involves mainly the environment specification, the definition of the agent initial position as well as its target position in the world (also called *goal*). By setting those parameters, a path-planning algorithm can be used to find a trajectory to be followed. However, in a real world, if we consider several persons (all in the same initial position) looking for achieving the same target position, each individual path followed will be different. Even if we have the same task, the strategy used for each one to reach his/her goal will depend on his/her physical constitution, personality, mood and reasoning.

In this paper we propose a path-planning approach based on boundary value problems to find paths between an initial and a target position in a dynamic environment. The paths found by our algorithm are smooth and variable, depending on the individual characteristics of each agent, which can be dynamically changed.

The paper is organized as follows. In Section 2 we presented some related work on path-planning techniques for virtual humans. Section 3 describes our path planner based on harmonic functions and Section 4 presents our main contribution, involving the extensions for the basic method. In Section 5 we deeply explain the way we implemented the method and in Section 6 we present our results. Finally, in Section 7 we present the conclusions and point out some future work.

## 2 Related Work

Motion planning methods have been largely studied by the robotics community. As in this paper our focus is on its use for simulating human beings behaviors while walking, we limit this Section scope for the works involving humanoids animation.

Lengyel *et al.* [3] have published one of the first articles on the subject of motion planning as a computer graphics problem. Their work presented a solution for the classical *Piano Movers* problem based on the use of standard graphics hardware to rasterize obstacles and generates the configuration space. The motion path produced by the planner is minimal with respect to the Manhattan distance metric and includes rotations and translations.

In order to generate more realistic results and allows its use in real-time applications, several authors proposed motion planning solutions based on two steps. In general, the first step is dedicated to define a valid path, while the second adapts this path in order to generate a more realistic movement. Kuffner [4] proposed a technique with the first step dedicated to the path-planning and the second to the path-following. The 3D scenario is projected in 2D and the humanoid treated as a disc, reducing the planning problem to a 2D problem.

Metoyer and Hodgings [5] proposed a similar technique also based on two steps. In their method, the characters have a pre-defined path to follows and

this path is smoothed and slightly changed to avoid collisions by using force fields.

The development of randomized path-finding algorithms, specially the PRM (Probabilistic Roadmaps) [6] and RTT (Rapidly-exploring Random Tree) [7], allow the use of large and most complexes configuration spaces, generating paths most efficiently. In this way, the challenge becomes more the generation of realistic movements than finding a valid path. Choi *et al.* [8] proposed the use of a captured movements library associated to the PRM to generate realistic movements in a static environment. Despite the fact the path maps should be generated in a pre-processing phase, the results are very realistic.

Thanks to the researches in robotics, the path-planning problem is almost solved. However, in the computer graphics domain, to find a natural and realistic way to move a character is as important as to find a path between two points. The most part of the works developed since now propose methods based on two separate phases. In the next sections we present our own proposal for generating realistic paths based on a single phase.

## 3 Harmonic Functions Path Planner

Whether it is a human being, a robot or a synthetic actor the action of moving from an initial position to a goal position in space consists of at least two stages: a planning stage when a path is devised; and an implementation stage when the path is followed by the moving agent. The first stage deals with a combination of concepts like efficiency, risk avoidance, computability, etc. To the second stage belongs the series of routines or corrections that the agent has to perform to adapt its motion when the predefined path cannot be followed due to unpredictable changes in the agent's surrounds, or in case of robotics, due to machine limitations.

In a seminal work in the field of robotics, Khatib [9] proposed a method that fuses these two stages in a very elegant way. He considered that instead of looking for a good path and trying to control the agent's movement around it a good planner should provide a potential field, or a force field (its gradient), that expanded the whole region of manoeuvre, producing a continuum of alternative paths. The potential field is devised to incorporate obstacles and goals, and should guide agent at all times indicating the best direction to follow. Its most straightforward implementation is a simple superposition of fictitious forces: obstacles forces that repel the agent to prevent collisions; and target forces that attract the agent. Such superposition is not always successful since for some environment configuration the agent can end up trapped in local minima before reaching the target.

Up to this date, the best way to produce a potential field that is free from local minima is through the numerical solution of a convenient partial differential equation with boundary conditions - a boundary value problem (BVP). The boundary conditions are central to the method indicating which regions in the environment are obstacles and which are targets.

The first proposal in this direction was made by Connolly and Grupen [10] and it is called the method of the harmonic functions. In their method the potential fields are the solutions of the Laplace's equation - whose solutions are called harmonic functions

$$\nabla^2 \, p(\mathbf{r}) \; = \; \sum_i \; \frac{\partial^2 p(\mathbf{r})}{\partial x_i^2} \; = \; 0 \tag{1}$$

where $\mathbf{r}$ is the environment coordinates. The Laplace's equation does not present local minima, and that is why it was chosen. They also proposed boundary conditions such that the potential should be one in the contours of the obstacles and zero in the region of the target. Setting up the value of the function in the boundaries is called a Dirichlet boundary condition in the language of a BVP.

The agent uses the gradient descent of this potential to determine the path that connects its current position to the target. As there is only a minimum defined in target position, it exists exactly one path from any point to the potential to the target. This method is formally complete, i.e., if there is a path that connects the agent position to the target it will be found. The resulting path is smooth and safe and it minimizes the collision probability with the obstacles.

## 4 Beyond Path Planner based on Harmonic Functions

Laplace's equation is not the only partial differential equation that generates functions without local minima. In [11], Trevisan *et al.* came up with a framework for exploratory navigation based on a family of potential field functions that does not possess local minima. The authors suggest the following equation

$$\nabla^2 \, p(\mathbf{r}) + \epsilon \mathbf{v}.\nabla p(\mathbf{r}) \; = \; 0 \tag{2}$$

for handling sparse environments, where $\mathbf{v}$ is a bias vector and $\epsilon$ is a scalar. The addition of the term $\epsilon \mathbf{v} \nabla p$ breaks the symmetry of vector field generated by Laplace's equation increasing the system performance in sparse environments during the exploration process.

The central contribution of this paper is to use the Equation 2 for generating different behaviors (illustrated in this work through the path followed by each agent) for several agents in a known environment. As discussed before, if the agent is controlled by a vector field produced by harmonic functions, it will always tend to follow a path that minimize the collision probability with the obstacles, i.e., in an indoor environment the agent will tend to follow a path equidistant to the walls, as shown in Figure 1(a). This behavior is not always adequate to simulate humanoid motion since it looks very stereotyped.

The adjustment of the vector $\mathbf{v}$ can produce a path close to the walls, as shown in Figures 1(b) and (c). The vector $\mathbf{v}$, also called *behavior vector*, can be seen as an external force field that counteract the natural tendency of agent moving away from the obstacles. The parameter $\epsilon$ can be understood as the

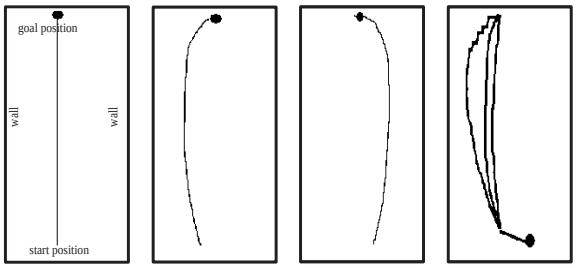


**Fig. 1.** Different paths followed by agents using Equation 2: (a) path produced by harmonic potential, i.e., with $\epsilon = 0$; (b) with $\epsilon = 0.8$ and $\mathbf{v} = (1, 0)$; (c) with $\epsilon = 0.8$ and $\mathbf{v} = (-1, 0)$; and using the same vector $\mathbf{v} = (1, 1)$ and different values to the parameter $\epsilon$ (0.4, 0.8, 1.2, 1.6, 2.0, 2.4 and 7.2).

*strength* or *influence* in following the direction defined by vector $\mathbf{v}$ instead of the direction produced by harmonic functions.

Figure 1(d) shows the results obtained in several experiments that use different $\epsilon$ and the same vector $\mathbf{v} = (1, 1)$. This flexibility allows to develop different and interesting behaviors to generate realistic humanoid motion during the navigation process. In our case, we simulated several agents with different $v$ and $\epsilon$ and put them into a known environment to perform a couple of navigation tasks.

## 5 Implementation

In this section, we present the global environment representation, the structure of the agents that act on the environment, as well as the mechanisms used to control each agent behavior.

### 5.1 Environment Global Map

The environment is represented by a *set* of homogenous meshes $\{m_k\}$, where each mesh $m_k$ is associated to a target $o_k$ and has $L_x \times L_y$ cells, denoted by $\{c^k_{i,j}\}$. Each cell $c^k_{i,j}$ corresponds to a squared region centered in environment coordinates $r = (r_i, r_j)$ and stores a particular potential value $p^k_{i,j}$. These maps are used by the harmonic path planner (see Section 3) to assist the agent to reach a specific target.

Each mesh $m_k$ has the potential values of its cells relaxed independently using the Equation 1. After the convergence, the map $m_k$ stores a potential field that is used to reach the target $o_k$. This procedure is performed before the simulation starts and we consider that the environment is surrounded by obstacles in order to delimit the agent navigation space.

### 5.2 Agent Local Map

Each agent $a_k$ has one map $am_k$ that stores the current local information about the environment obtained by its sensors. This map is centered in the current agent position and represents a small fraction of the global map, nearly 10% of the total area covered by the global map.

The map $am_k$ has $l_x^k \times l_y^k$ cells, denoted by $\{ac_{i,j}^k\}$ and can be divided in three regions: the update zone (*u-zone*); the free zone (*f-zone*) and the border zone (*b-zone*), as shown in Figure 2(a). In a similar way, each cell corresponds to a squared region centered in environment coordinates $r = (r_i, r_j)$ and stores a particular potential value $ap_{i,j}^k$.

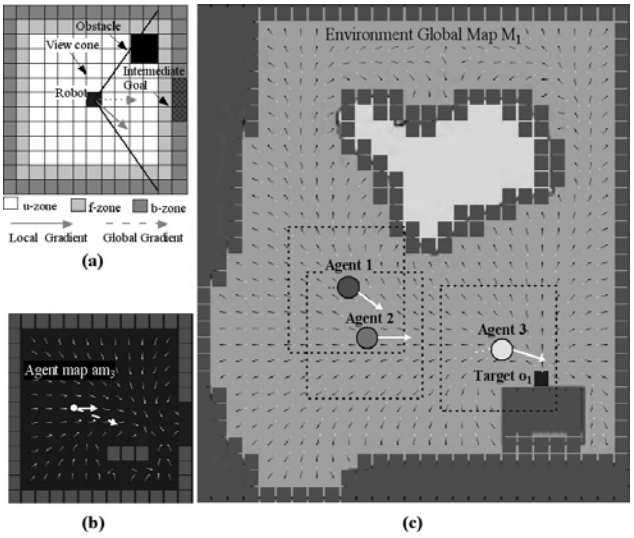


**Fig. 2.** (a)Agent Local Map. The white, light gray and dark gray cells comprise the update, free and border zones, respectively. (b,c)Agents acting in the real environment. Each agent senses the environment, updates its local map (b) and navigates towards the target $o_1$ (c).

The area associated to each agent map cell is smaller than the area associated to the global map cell. The main reason is that the agent map is used to produce a refined motion, hence, the smaller cell size the better the quality of motion; while the global map is used only to assist the long-term agent navigation.

### 5.3 Updating Local Maps using Global Maps

Each agent $a_k$ has a well determined goal $o_{goal(k)}$ (the function *goal* maps the agent number $k$ into its current target number. In this description, we will consider that each agent must reach only one target. The extension to multiple

targets is straightforward and will be commented in Section 5.5), a particular vector $\mathbf{v}_k$, that controls its behavior, and a $\epsilon_k$ that determines the influence of $\mathbf{v}_k$. The same goal, $\mathbf{v}$ and $\epsilon$ can be designated to several agents.

When the agent $a_k$ navigates the environment, it uses its sensors to perceive the environment and to update its local map with the information about the obstacles and other agents. The agent sensors set a view cone with aperture $\alpha$.

Figure 2(b) sketches a particular instance of the agent local map. The *u-zone* cells $\{ac^k_{i,j}\}$, inside the view cone, with obstacles or agents have their potential value set to 1. Obstacles are not considered in the *u-zone* out of the view cone. This procedure assures that dynamic or static obstacles behind the agent do not interfere in its future motion.

The agent $a_k$ calculates the global descent gradient on the cell in the global map $m_{goal(k)}$ containing its current position. The gradient direction is used to generate an intermediate goal in the border of the local map setting the potential values to 0 of a couple of *b-zone* cells. While the other *b-zone* cells are considered as obstacles having their potential values set to 1. In Figure 2(c), each agent calculates the global gradient in order to project an intermediate goal in its local map. As the agent local map is delimited by obstacles, the agent is pulled towards the intermediate goal using the direction of its local gradient. The intermediate goal helps the agent $a_k$ to reach its target $o_{goal(k)}$ while allowing it to produce its particular motion.

In some cases, the target $o_{goal(k)}$ is inside both the view cone and the *u-zone*, and consequently, the local map cells associated are set to 0. The intermediate goal is always projected even if the target is mapped onto the *u-zone* otherwise the robot can easily get trapped because the robot would be taking into consideration only the local information about the environment, in a same way as traditional potential fields [9].

The *f-zone* cells are always considered free of obstacles, even when there are obstacles there. The absence of this zone may close the connexion between the current agent cell and the intermediate goal due to the mapping of obstacles in front of intermediate goal. When this occurs, the robot gets lost because there is no information coming from the intermediate goal to produce a path to reach it. The *f-zone* cells handle the situation permitting always that the information about the goal is propagated to the cells associated to the agent position.

After the sensing and mapping steps, the agent updates the potential value of its map cell using a discrete version of Equation 2,

$$ap^k_{i,j} = \frac{1}{4}(ap^k_{i-1,j} + ap^k_{i+1,j} + ap^k_{i,j-1} + ap^k_{i,j+1}) + \frac{\epsilon^k}{8}((ap^k_{i+1,j} - ap^k_{i-1,j})v^k_x + (ap^k_{i,j+1} - ap^k_{i,j-1})v^k_y) \quad (3)$$

where $\mathbf{v}^k = (v^k_x, v^k_y)$ is the vector that controls the behavior of agent $a_k$ and $\epsilon^k \in [-2, +2]$ and represents the *influence* of vector $\mathbf{v}^k$. The local potential is partially relaxed [12] and the agent calculates the gradient descent of its position

in the local map $am_k$ by

$$\mathbf{dgrad}^k = \Big((ac^k_{p_x+1,p_y} - ac^k_{p_x-1,p_y})/2, (ac^k_{p_x,p_y+1} - ac^k_{p_x,p_y-1})/2\Big)$$

where $p_x = \lceil l^k_x/2 \rceil$ and $p_y = \lceil l^k_y/2 \rceil$, and it follows the direction $\theta^k$ calculated by $\theta^k = \arctan(dgrad^k_x, dgrad^k_y)$ where $\arctan(.,.)$ is the inverse tangent taken in the interval $[-\pi, +\pi]$.

### 5.4 Characterizing the Agent Behavior

In the real world, even if several people have the same goal, the strategy used for each one to reach it will depend on different factors as: physical constitution, personality, mood, and reasoning. In Figure 1 we shown we can simulate different behaviors by setting both the behavior vector $v$ and $\epsilon$ differently for each agent. In this first example we kept the variables constant during the animation, however we can produce more interesting behaviors dynamically changing vector $\mathbf{v}$ and $\epsilon$. For instance, the vector $\mathbf{v}$ can be controlled by a function defined by the user, as in Figure 3. Even with this new complex behavior, which simulates a *drunk* agent, the resulting potential guarantees that the robot reaches safely the target.

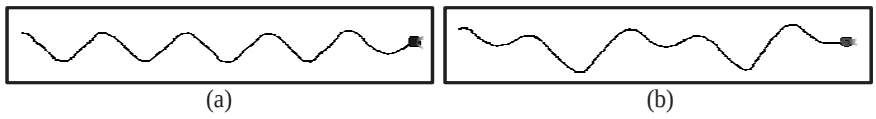


**Fig. 3.** Paths followed by agents using different equations that control the behavior vector $\mathbf{v}$: (a) $\mathbf{v} = (1, \sin(\omega * t))$; (b) $\mathbf{v} = (1, \sin(\omega * t) + \sin(\omega/2 * t))$, with $\omega = \pi/18$ and $t$ the current simulation step.

We can change $\mathbf{v}$ in a regular periodic fashion, as shown above, but it does not need always to be the case. We can consider an agent that randomly changes its behavior vector. Each new value of $\mathbf{v}$ is kept constant during an also random time interval.

### 5.5 Algorithm

In this section we present the algorithm that implements the concepts shown before and produce the humanoids simulation.

1. computes all the environment global maps (one for each possible goal $o_k$)
2. for each agent $a_k$, defines the behavior vector $\mathbf{v}_k$ and $\epsilon_k$. Each variable can be either static or dynamic. If a variable is chosen to be dynamic then the function that controls it must be specified.

3. for each agent $a_k$ do (asynchronously)
   (a) reads its sensors in order to detect static and dynamic obstacles
   (b) updates its map with local information about the obstacles and other agents
   (c) computes the global gradient descent and generates the intermediate goal
   (d) updates the potential field
   (e) computes the local gradient descent and follows the gradient direction
   (f) while not reaching the target $o_{goal(k)}$ repeat the steps from $(a)$ to $(f)$, otherwise stops moving

The first two steps are performed in a pre-process phase. In relation to the step 3, each agent executes independent and asynchronously the actions from $(a)$ to $(f)$. This algorithm considers each agent must reach only one target. However, the agent can be in charge of reaching several targets orderly. In this case, the step $(f)$ must be changed to

(f) while not reaching the target $o_{goal^i(k)}$ repeat the steps from $(a)$ to $(f)$, otherwise if $goal^i(k) = goal^{last}(k)$ then stops moving. Else repeats the process with the next target $o_{goal^{i+1}(k)}$

## 6 Results

In order to illustrate the potentialities of our path-planning approach we made some experiments considering a realistic situation. Taking into account the scenario described bellow, we have induced some agents' behaviors to verify some considerations made before, as: how to accomplish the same task in different ways; or how different agents avoid collisions, for example. In another set of tests we have ran the algorithm considering a variable number of agents with random objectives, behaviors and velocities. Our goal with these experiments was to verify the motion diversity. Finally, we made some considerations about performance.

We consider a small park in a town (see Figure 4). It has five accesses, a lake in the middle and a popcorn-cart in the south. Characters in the simulation can simply cross the park or stop to buy a popcorn bag and continue their walking. It is a quite familiar real scenario; the large open area makes easy and clear the simulation of different agents behaviors that will not be constrained by an excessive amount of obstacles; by simulating a group of agents walking in the park it is easy to verify the collision avoidance with dynamic obstacles (here represented as other agents).

The set up for this scenario involves the statement of six possible goals, one for each park access and another in front of the popcorn-cart. We will need to compute 6 environment global maps. In our tests, we used a matrix with 60x60 cells to represent global maps and a matrix with 11x11 cells for the agent local maps.

The first situation induced by us consists in simulating the behavior of 4 agents while accomplishing the same task. The agents are initially disposed somewhere in the park access west and their task consists on go to the popcorn-cart and after, to quit the park by the access north. Figure 4 illustrates the results of animation. Each agent accomplishes its task individually without the intervention of the others. The small square specifies the moment where the agent 3 changes its behavior vector $\mathbf{v}$.

Figure 4(b) shows the same task of Figure 4(a), but in this case all the agents are moving at the same time, therefore they compete for the targets. The paths drawn in these two figures are slightly different and these differences are duo the collision detection and avoidance between the agents.


(a) (b) (c)

**Fig. 4.** Four agents individually accomplishing the same task (a) and accomplishing the same task concurrently (b). Agent 1: $\mathbf{v} = (-0.707, -0.707)$, $\epsilon = 0.8$ and $step = 0.6$ cells per frame; agent 2: $\mathbf{v} = (0.707, 0.707)$, $\epsilon = 0.8$ and $step = 0.5$ cells per frame; agent 3: initially $\mathbf{v} = (\sin(\omega * t), 1)$, changing to $\mathbf{v} = (0.707, 0.707)$ after some time, $\epsilon = 0.8$ and $step = 0.35$ cells per frame; agent 4: $\epsilon = 0$, and $step = 0.46$ cells per frame.(c) Simulation of a set of 12 agents walking around the park with random behaviors.

Figure 5 shows two frames of the animation of two agents. One agent walks from the north to the south while the other one walks from the south to the north. Using our algorithm we automatically avoid the collision between the two agents, since each agent is considered as a dynamic obstacle by the other. However, the final path definition can be more or less natural, depending on the parameters definition. In the sequence presented on Figure 5(a), we set $\epsilon$ as 0. In this way, the behavior vector $\mathbf{v}$ is not considered. For the animation shown in Figure 5(b), both agents begins the animation with $\epsilon = 0.0$. When the proximity is detected, the behavior vector $\mathbf{v}$ of each agent is oriented orthogonally to the collision direction, forcing the movement to its right direction. At the same time, the $\epsilon$ becomes equal to 0.6.

Finally, we generated some animation sequences without searching to reproduce any specific behavior. In those sequences we used 12 agents, $\epsilon = 0.8$ for all agents and the components of $\mathbf{v}$ randomly varying between -1 and 1. The

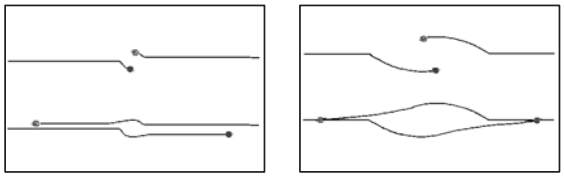


**Fig. 5.** Two collision avoidance animation sequences produced with different values for the behavior vector and $\epsilon$.

agents step size are also randomly defined between 0.3 and 1.0 cells per frame. The initial and final positions for the agents are arbitrarily chosen. The agents can begin its movement from any valid position in the environment and its goal is one of the 6 possible target positions described before. Figure 4(c) shows a frame of one of the animation generated by us.

## 7 Conclusions and Future Work

This article presents a new approach for generating pedestrian behavior using path planning based on numerical solution of boundary value problems. We demonstrate that the correct adjustment of behavior vector and the parameter $\epsilon$, that determines the vector influence, can produce interesting behaviors, as illustrated in Figures 1 and 3. These behaviors can be interchanged to produce complex motions, as shown in Figure 4, oriented to the agent personality. In this work, we do not implement the agent personality. This step is actually in progress and will be shown in our future submissions.

The guiding potential of Equation 2 is free of local minima what constitutes a great advantage when compared to the traditional potential fields. Furthermore, the method proposed is formally complete and generates smooth and safe paths that can be directly used in mobile robots. The local information gathered by agent sensors allows treating the dynamic obstacles, as other agents navigating in the environment.

We handle the usual costs associated to BVP calculations by using small local maps, instead of a large map that cover all the environment, for each agent. This permits to have several agents acting in the environments while keeping an acceptable running time. Even with only local information about the environment, the intermediate goals computed from the environment maps add global information about the agent target in order to treat conveniently local minima and to allow the agent to reach its target.

Another drawback is that the potential gets flat far from the target position due to numerical precision. In these regions, the gradient is very small to provide useful information to guide the robot. In this case, the robot can easily get lost.

We have successfully overcome this problem by using intermediate goals in the flat region.

In the future, we intend: to test different path planners to minimize the computational cost associated to the environment global map; to develop an architecture to be implemented into the GPU to reduce the potential time computation; and to develop an efficient data structure to compact the environment information, such as quadtree, and an efficient algorithm to access this information in real-time.

## Acknowledgments

We would like to thank FAPERGS and CNPq for financial support and Renato Oliveira for helping with the figures.

## References

1. Shao, W., Terzopoulos, D.: Autonomous pedestrians. In: ACM SIGGRAPH/Eurograph symposium on Computer Animation. (2005) 19–28
2. Torres, J.A., Nedel, L.P., Bordini, R.H.: Using the bdi architecture to produce autonomous characters in virtual worlds. In: Intelligent Virtual Agents. Volume 2792 of Lecture Notes in Artificial Intelligence., Springer Verlag (2003) 197–201
3. Lengyel, J., Reichert, M., Donald, B.R., Greenberg, D.P.: Real-time robot motion planning using rasterizing computer graphics hardware. Computer Graphics **24**(4) (1990) 327–335
4. James J. Kuffner, J.: Goal-directed navigation for animated characters using real-time path planning and control. In: International Workshop on Modelling and Motion Capture Techniques for Virtual Environments, London, UK, Springer-Verlag (1998) 171–186
5. Metoyer, R.A., Hodgins, J.K.: Reactive pedestrian path following from examples. The Visual Computer **20**(10) (2004) 635–649
6. Kavraki, L., Svestka, P., Latombe, J.C., Overmars, M.: Probabilistic roadmaps for path planning in high-dimensional configuration space. IEEE Transactions on Robotics and Automation **12**(4) (1996) 566–580
7. LaValle, S.: Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Dept., Iowa State University (1998)
8. Choi, M.G., Lee, J., Shin, S.Y.: Planning biped locomotion using motion capture data and probabilistic roadmaps. ACM Trans. Graph. **22**(2) (2003) 182–203
9. Khatib, O.: Commande dynamique dans l'espace opérational des robots manipulaters en présence d'obstacles. PhD thesis, École Nationale Supérieure de l'Aéronatique et de l'Espace, France (1980)
10. Connolly, C., Grupen, R.: On the applications of harmonic functions to robotics. International Journal of Robotic Systems **10** (1993) 931–946
11. Trevisan, M., Idiart, M.A., Prestes, E., Engel, P.M.: Exploratory navigation based on dynamic boundary value problems. accepted for publication in Journal of Intelligent and Robotic Systems (2006)
12. Prestes, E., Engel, P.M., Trevisan, M., Idiart, M.A.: Exploration method using harmonic functions. Robotics and Autonomous Systems **40**(1) (2002) 25–42