The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

Encoding User Motion Preferences in Harmonic Function Path Planning

Giles D'Silva and Manfred Huber

Abstract— **Humans have unique motion preferences when pursuing a given task. These motion preferences could be expressed as moving in a straight line, following the wall, avoiding sharp turns, avoiding damp surfaces or choosing the shortest path. While it would be very useful for a range of applications to allow robot systems or artificial agents to generate paths with similar specific characteristics, it is generally very difficult to capture and reproduce them from observed information since user trajectories can not be easily generalized. To address this, this paper introduces an approach that modifies a harmonic function path planner to model the user's motion preferences as parameters which could then be used to generate new paths in similar environments without the risk of collisions or incorrect paths. Given a small set of user-specific trajectories and starting from an initial, generic parameter configuration, this approach incrementally minimizes the difference between the direction of the user trajectory segments and the gradient of the parametric harmonic function by modifying its underlying parameters, thus capturing the trajectory preferences. Subsequently, these parameters could be transferred to new, locally similar environments and used to generate new paths. The use of harmonic function parameters to represent the user preferences not only facilitates customization of the path planner but also assures that the customized planner remains complete and correct.**

I. INTRODUCTION

Human motions to achieve a similar task are usually distinctive and particular to their individualistic preferences. Being able to capture such preferences and generate corresponding paths would be very useful for a range of applications of robot systems or artificial agents, including assistive wheelchairs, social robots, intelligent game characters, or characters for social agent simulations, by allowing them to better address the user's implicit objectives and by generating more realistic and diverse navigation behavior. To address this need for more realistic or customized trajectories, path planners that optimize global performance metrics [1] and systems that generate more realistic human paths have been studied [2], [3]. However, while these systems generate modified trajectories, the criteria used do generally not represent individual preferences (but rather reflect global performance metrics or effects imposed by the system dynamics) and are designed manually rather than extracted autonomously from examples of the navigation preferences.

One way to allow for broader personalization of the autonomous generation of paths would be to pre-program the path planner for each individualistic motion behavior. However this would be a highly complex task given the large number of possible preferences for each user. A better approach would be to model and autonomously learn these preferences so that they can be automatically generated from user information and be applied to new environments.

In this paper, a Harmonic function path planner was selected as the basis for an approach to path personalization because of its robustness, completeness, ability to exhibit different useful modes of behavior, and rapid computation [4]. The boundary conditions used by the harmonic function motion planner assure that the agent maintains safe distances from obstacles. To achieve the personalization, motion preferences, represented through a set of trajectories extracted from the system whose preferences are to be captured, are modeled in the form of weight modifications that are then used in the path planning process to generate paths with similar "characteristics". The weight modifications are learned automatically by minimizing the directional error between the user trajectories and the computed gradient, allowing the planner to optimize its policy. Since the influence of different weights on the error decreases exponentially with distance, a local assumption of weight influence on the error is used to make the error minimization tractable.

The remainder of the paper first discusses related work before the methodology and the algorithm are introduced and evaluated on a number of path modification examples. To simplify the notation, the derivation throughout this paper is illustrated for a two dimensional path planning problem. However, the approach can easily be extended into higher dimensional configuration spaces, allowing it to be applied to arbitrary path planning domains.

A. Related Work

Harmonic functions have been used by a number of researchers to generate smooth, collision-free paths. Among these, a number of approaches have also addressed the question of modifying the paths that are generated by either modifying parameters within the function representation or by altering the boundary conditions under which the function is relaxed. The boundary conditions selected can give rise to different harmonic functions and the resulting paths vary with respect to safety. In [5] a new harmonic function was formulated by taking a linear combination of two common boundary conditions to generate minimum time paths from every point in the environment to the goal. However, this approach towards path modification has limitations with respect to the number of different paths that can be generated. Path modification techniques have also been proposed in [6] [7], where a hybrid planner combines harmonic functions with probabilistic roadmaps to generate minimal paths.

G. D'Silva is with the Department of Computer Science and Engineering at the University of Texas at Arlington, TX 76019, giles.dsilva@mavs.uta.edu

M. Huber is with the Department of Computer Science and Engineering at the University of Texas at Arlington, TX 76019 huber@cse.uta.edu

To achieve more flexible customization, the research by Coelho [1] addresses how the conductances in a 2D resistive grid of a harmonic function-based motion controller can be adjusted to optimize a user-specified performance criterion. The methodology used to update these conductances over a grid equivalent circuit uses a policy iteration algorithm in which the control policy is repeatedly modified until the optimal policy generates a gradient which optimizes the given performance index. The paths generated are safe and the capability to reach its goal (correctness) is preserved at every step of the policy iteration. However, this approach is aimed at global optimization of the overall potential function rather than at capturing locally expressible trajectory preferences from user trajectory data. Trevisan [8] follows a similar path to the modification of paths and shows that a number of potential functions exist that do not possess local minima based on which Fabio [9] proposed adding external force fields to counteract the natural tendency of the agents to follow the direction provided by the planner. This approach suggests creating a potential field that includes a bias vector that encodes separate weight changes for each dimension and represents an external force field which captures different agent preferences. However, the approach does not provide any means to automatically extract the desired external force field characteristics and, through the choice of the bias vector, introduces limitations in terms of the types of customizations that can be achieved. The approach presented in this paper alleviates these limitations by allowing weights to be modified independently and by providing a framework that allows for the automatic derivation of local weight changes.

II. METHODOLOGY

In this research, a modified harmonic function path planner is used to generate customized policies. The distinctive paths which the planner generates are here a direct result of the preferences provided by a set of user specified trajectories. Fig.1 shows an overview of the learning and evaluation process used to extract the preferences from the sample trajectories and translate them into weight parameters for the harmonic function path planner.

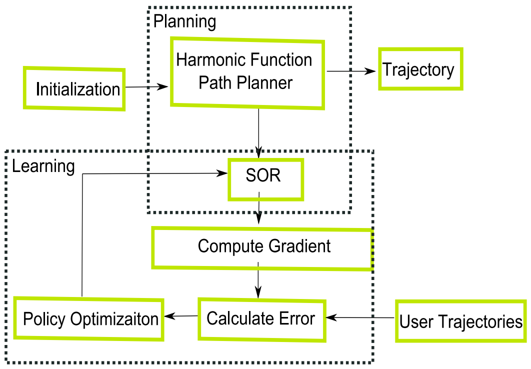

Fig. 1. Desired path generation overview

In this approach, weights are learned using an iterative process in which weights are modified and the harmonic potential is re-relaxed to reflect changes in the computed gradient. In every iteration the error is computed to determine whether the user trajectories are successfully approximated and if the error is below a threshold the process terminates.

A. Harmonic Functions

Harmonic functions for path planning were first proposed by Connolly and Grupen [4] and have since been widely used. Harmonic functions generate legal paths from any point in the configuration space, if they exist, and the path taken is a gradient descent on the potential field,, where ϕ is defined over a domain $\Omega \subset \Re^n$ and satisfies Laplace's equation:

$$\nabla^2\phi = \sum_{i=1}^{n} \frac{\partial^2\phi}{\partial x_i^2} = 0$$

The control policy generated by a harmonic function path planner is generic in nature and paths that are stereotyped by user preferences remain unexplored. To address this, the harmonic function used here utilizes a grid-based potential where the value at a grid cell is computed as a weighted average over its neighbors. This always assures the absence of local extrema and thus ensures the correctness of the paths generated. Let $\phi(x_i, y_j)$ be the potential value of the harmonic function for a cell located at x_i, y_j and $w_n(x_i, y_j)$ be the weight that connects the cell to its surrounding neighbors. The equation for the modified harmonic function in two dimensions can then be represented as

$$\begin{aligned}\phi(x_i, y_j) = (&\phi(x_{i-1}, y_j) * w_1(x_i, y_j) \\ &+\phi(x_{i+1}, y_j) * w_2(x_i, y_j) \\ &+\phi(x_i, y_{j+1}) * w_3(x_i, y_j) \\ &+\phi(x_i, y_{j-1}) * w_4(x_i, y_j))\end{aligned} \tag{1}$$

where,

$$w_1 + w_2 + w_3 + w_4 = 1$$

1) Weight Representation: The interdependence of the weights in Equation (1) can lead to constraints that have to be obeyed when optimizing the potential field. To alleviate these constraints, the weights are modeled here by functions with independent parameters which are used in later sections to capture the preferences of user motion trajectories. In particular, the four weights at each node are represented by the three independent parameters $g0, g1, g2$ where $g0$ represents a balancing factor between the X and Y dimensions while parameters $g1$ and $g2$ model the bias between neighbors in the X and Y dimension, respectively. Let w_1, w_2, w_3 and w_4 be the four weights of a node connecting it to each of its neighbors, then the harmonic function in parametric form at x, y can be represented as

$$\begin{aligned}\phi(x, y) = &g0[g1\phi(x-1, y) + \phi(x+1, y)(1-g1)] \\ &+(1-g0)[g2\phi(x, y-1) + \phi(x, y+1)(1-g2)]\end{aligned} \tag{2}$$

where,

$$0 \leq g0, g1, g2 \leq 1$$

To remove the limit constraints, the three parameters are further encoded by three sigmoid functions $g_i = \sigma(t_i) =$

$\frac{1}{1+e^{-t_i}}$ in order to obtain independent, unconstrained parameters, t_i, which are updated by the algorithm. The value of this sigmoid function converges to 0 as the sigmoid parameter $t_i \rightarrow -\infty$ and to 1 as $t_i \rightarrow \infty$. Given these parameter transformations, all updates made to the parameters are independent of each other and do not result in a conflict when minimizing the error function.

2) Potential Field Relaxation: Given a set of values for the weight parameters, successive over-relaxation (SOR) is used to solve the homogeneous system of linear equations (see Equation (1)) representing the solution to the harmonic function. This technique is based on the Gauss-Seidel iterative method but converges much faster by accelerating the approximation through a relaxation factor c, where $0 \leq c \leq 2$. The iteration is terminated if the magnitude of the update falls below a specified residual value.

3) Gradient Computation: The policy generated by the harmonic function path planner is retraced by following the negative gradient of the potential field. The gradient provides the direction of motion from every point in the environment. The gradient $\vec{\pi}_{x,y}$ at a particular point (x, y) on the grid, is computed as the potential difference between its neighbors in each dimension divided by twice the square of their distance. Given the neighboring potentials $\phi(x_{i-1}, y_j); \phi(x_{i+1}, y_j)$ and $\phi(x_i, y_{j+1}); \phi(x_i, y_{j-1})$ in the X and Y dimensions, respectively, the gradient $\vec{\pi}_{x,y}$ is

$$\vec{\pi}_{x,y} = (\frac{\phi(x_{i-1}, y_j) - \phi(x_{i+1}, y_j)}{2\Delta^2}, \frac{\phi(x_i, y_{j-1}) - \phi(x_i, y_{j+1})}{2\Delta^2})$$

where Δ is the internodal distance between cells

III. Modifying Weights To Capture Trajectory Preferences

To modify the control policy to match a user's trajectory, the symmetry between weights needs to be relaxed. A higher weight value here indicates a stronger connection and vice versa. Modifying the connection weights between the left and right neighbors over the entire local area of a hallway, shown in Fig.2, produces a change in trajectory from the one in Fig.2A to the one in Fig.2B. The change is produced by allowing the left wall to exert a greater repulsive force, causing the agent to drive along the right side.

The policy generated by the modified planner still maintains all the characteristics of a good path planner. However, modeling the effects of weight modifications on the gradient can become very complex in a dense environment. E.g. Fig.2C shows that the control policy for the same weight modification in a crowded hallway is significantly different and can not be easily understood. As a consequence, modifications can not easily be applied manually but rather an automatic weight modification mechanism is needed.

IV. Learning Weights

Weight modifications can be made to relatively small regions of the grid to better predict the behavior of the agent in that region. Weight modifications in small local regions, however, may influence other regions by a magnitude which is hard to formalize. Weights should thus be modified such that the global effect is taken into account.

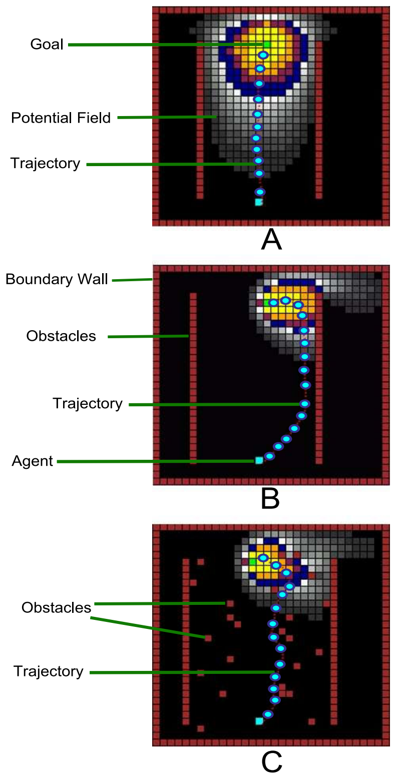

Fig. 2. Generic trajectory

In the preference learning phase of the approach presented here, the weights are adjusted iteratively until the control policy closely approximates the reference policy provided by a set of user trajectories. To do this, the planner tries to minimize an error function, e, which represents the sum of the angle differences between the directions of the reference trajectories and the corresponding gradient directions of the potential along the user paths.

$$e = \sum_{node \in path} (1 - cos(\vec{\pi}, \vec{\nabla} P))^3 \qquad (3)$$

The reference policy $\vec{\nabla} P$ is extracted from user trajectories and $\vec{\pi}$ is the gradient computed over the current potential field. The third power is considered in order to magnify large errors. Using this error function, gradient descent is used to learn a set of weights which lead to a navigation policy which deviates minimally from the user trajectories. The learning phase terminates when the value of the error function e is reduced below a specified threshold.

A. Policy Optimization

Algorithm 1 adjusts the set of weights W, to generate a policy $\vec{\pi}$, that approximates a user's trajectory. This algorithm modifies weights to minimize the error function in Equation (3) by computing the derivative of the error with respect to the sigmoid parameters. Weights are updated and the new gradient is computed. This process continues until the error lies below a specified threshold ϵ.

```
**Input**: User trajectory
**Output**: Gradient matching user path
Compute the direction vectors for each node along the user's trajectory Compute $\vec{\pi}$ = - $\vec{\nabla}\phi$, given weights $W$
Compute error $e$
while (approximation error $e$ is above a threshold $\epsilon$) do
    Compute $\frac{\partial e}{\partial \phi}$ for all nodes using error propagation
    Compute the gradient $\vec{\nabla} e = \frac{\partial e}{\partial \vec{t}} = \sum \frac{\partial e}{\partial \phi} \cdot \frac{\partial \phi}{\partial \vec{t}}$
    Update sigmoid parameters $\vec{t} = \vec{t} + \alpha \vec{\nabla} e$
    Adjust weights $W$
    Compute $\vec{\pi}$ = - $\vec{\nabla}\phi$, given weights $W$
    Compute error $e$
end
```

Algorithm 1: Weight Modification

Weight adjustments are computed by taking the derivative of the error with respect to each of the sigmoid parameters (t_{g0}, t_{g1}, t_{g2}) and propagating the corresponding update back into a modified weight set.

In order to compute the derivative, the chain rule is applied as shown in Equation (4) to break the calculation into addressable components.

$$\vec{\nabla} e = \frac{\partial e}{\partial \vec{t}} = \sum \frac{\partial e}{\partial \vec{\pi}} \cdot \frac{\partial \vec{\pi}}{\partial \phi} \cdot \frac{\partial \phi}{\partial w} \cdot \frac{\partial w}{\partial p} \cdot \frac{\partial p}{\partial \vec{t}} \tag{4}$$

where,

$$\vec{t} = (t_{g0}, t_{g1}, t_{g2})^T$$

1) Locality Assumption: The amount of influence a node weight has on the error function depends on its distance from a path node. While all node weights of the local environment have an influence on the node's potential, their effect decrease exponentially with distance. To make the gradient calculation feasible, a local approximation template is designed which defines the weights that directly influence the potential at a given node. This template is placed over each node to find the nodes whose weights have direct influence on the given node's potential and is a local approximation of the global effect of weights on the error at a node. The template (shown in Fig.3) is designed such that it covers all node weights which are less than two steps away from the selected node. The potential of nodes outside the template are assumed to be fixed and thus its weights have no direct influence on the selected node.

Using this template, the change in error produced by these weights can be calculated by taking the derivative of the error with respect to each weight covered by the template. Each weight directly or indirectly influences the potential at more than one node and thus the derivative of the potential ϕ_q at cell q with respect to the weights can be expressed as

$$\frac{\partial \phi_q}{\partial W} = \sum_{q_i \in N(q), 1 \leq j \leq 4} \frac{\partial \phi_q}{\partial w_j(q_i)} \tag{5}$$

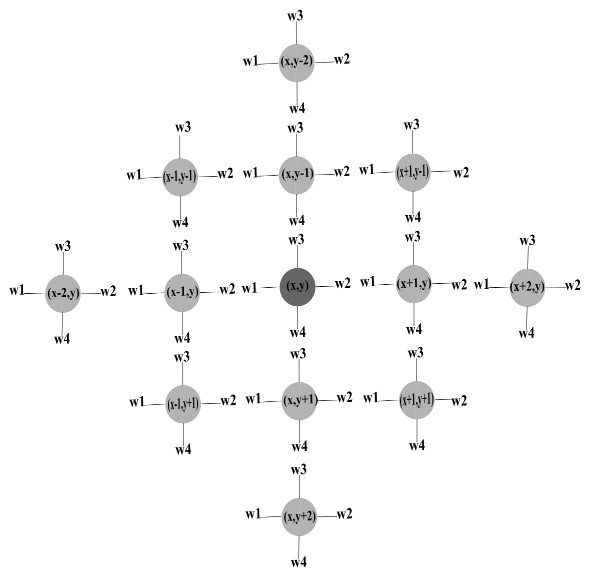

Fig. 3. Weight influence template

where,

$$N(q) = \{q_i | q_i \leq \ 2 \ steps \ from \ q\}$$

Node q is the node on which the template is placed and $N(q)$ is the neighborhood defined by the template.

2) Potential Error Propagation: While it is reasonable to assume that a weight's influence on the potential value is local, the influence of potential changes on the error function along a particular trajectory does not warrant such a local assumption. In particular, in order to minimize the error along a trajectory it might be necessary to adjust potential values in cells at a significant distance. An algorithm similar to the backpropagation is used here to propagate the error outward from the nodes whose directional error is known. These nodes act similar to the hidden nodes of a feed forward neural network and the error is propagated sequentially backward one step at a time to avoid recursive dependencies. Manhattan distance is used to label nodes to represent the propagation structure shown in Fig.4.

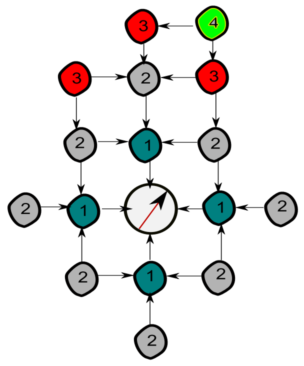

Fig. 4. Network for error propagation

In the first iteration every neighbor of a path node gets labeled 1 and incrementally all other nodes are labeled. Nodes that are marked as the goal or an obstacle have no error and so the derivative of its potential with respect to its weights is always zero. The potential error at a node is calculated by finding the derivative of the error with respect to its potential. The derivative for each node is initialized to zero and, as seen in Algorithm 1, the derivative of the error with respect to the potential at nodes labeled 1 is first computed. These nodes are neighbors to at least one of the nodes that lie along the path and thus have a directional error. The change in potential at a node q_i labeled 1 has a direct influence on the directional error of its neighbors and the derivative of the error function with respect to its potential ϕ_{q_i} for $label(q_i) = 1$ is calculated as follows: [1],

$$\begin{aligned}\frac{\partial e}{\partial \phi_{q_i}} &= \sum_{q_j \in M(\phi_{q_i})} \frac{\partial (1 - cos(\vec{\pi}, \vec{\nabla} P))^3}{\partial \phi_{q_j}} \\ &= \sum_{q_j \in M(\phi_{q_i})} 3(1 - cos(\vec{\pi}, \vec{\nabla} P))^2 . \frac{\partial}{\partial \phi_{q_j}} [\frac{\vec{\pi}.\vec{\nabla} P}{|\vec{\pi}||\vec{\nabla} P|}]\end{aligned} \tag{6}$$

Where $M(\phi_{q_i})$ are the immediate successors of q_i under the propagation structure (Fig.4). $\vec{\nabla} P$ is the optimal control policy we are trying to match, and $\vec{\pi}$ is the control policy we modify to achieve a match. Following this,

$$\frac{\partial}{\partial \phi_{q_j}} [\frac{\vec{\pi}.\vec{\nabla} P}{|\vec{\pi}||\vec{\nabla} P|}] = \frac{1}{|\vec{\pi}||\vec{\nabla} P|} [\frac{\partial \vec{\pi}}{\partial \phi_{q_j}} . \vec{\nabla} P - \frac{\vec{\pi}.\vec{\nabla} P}{|\vec{\pi}|^2} (\frac{\partial \vec{\pi}}{\partial \phi_{q_j}} . \vec{\pi})]$$

and given that $\vec{\pi} = -\vec{\nabla}\phi$, $\vec{\pi}$ at node q_i can be expressed as

$$\vec{\pi}_k = \frac{\phi_{q_i - k} - \phi_{q_i + k}}{2\Delta^2}$$

$$\frac{\partial \vec{\pi}_k}{\partial \phi_{q_j}} = \frac{1}{2\Delta^2} [\frac{\partial \phi_{q_i - k}}{\partial \phi_{q_j}} - \frac{\partial \phi_{q_i + k}}{\partial \phi_{q_j}}]$$

where $\vec{\pi}_k$ is the k^{th} dimension of the policy vector, $q_i -_k$ and $q_i +_k$ are the left and right neighbors of node q_i in the k^{th} dimension of the grid, and Δ is the internodal distance.

The derivative of the error with respect to the potential ϕ_{q_i}, for nodes q_i with $label(q_i) \geq 2$ can subsequently be computed by propagating the error according to

$$\frac{\partial e}{\partial \phi_{q_i}} = \sum_{q_j \in M(\phi_{q_i})} \eta_{q_j} . \frac{\partial \phi_{q_j}}{\partial \phi_{q_i}}$$

where,

$$\eta_{q_j} = \frac{\partial e}{\partial \phi_{q_j}}$$

3) Computing Parameter Adjustments: Given the calculation of the derivative of the error with respect to the potential described in the previous sections, it remains to determine the parameter space gradient of the weights in order to complete the gradient calculation step in Algorithm 1 and to allow for the weight update. From Equation (4) the parameter space gradient $\frac{\partial \phi}{\partial t}$ can be reduced to Equation (7):

$$\frac{\partial \phi}{\partial t} = \frac{\partial \phi}{\partial \vec{w}} \frac{\partial \vec{w}}{\partial \vec{p}} \frac{\partial \vec{p}}{\partial \vec{t}} \tag{7}$$

where,

$$\vec{w} = (w_1, w_2, w_3, w_4)^T$$
$$\vec{p} = (g0, g1, g2)^T$$
$$\vec{t} = (t_{g0}, t_{g1}, t_{g2})^T$$

Since $\phi_q = \phi(x, y)$ is computed as a weighted average of its neighbors, its derivative with respect to the weights in the local template can be computed. For this, first the derivative for the direct weights of the node at the center of the template (see Fig.3) is computed from Equation (1). Then the derivative with respect to weights of nodes at distance greater than 0 in the template can be computed indirectly. Let $w_i(q_j)$ be weight i of the node located at q_j where q_j is at a distance d from q in the local template. Then the derivative can be expressed as

$$\frac{\partial \phi_q}{\partial w_i(q_j)} = \frac{\partial \phi_q}{\partial \phi_{q_j}} \frac{\partial \phi_{q_j}}{\partial w_i(q_j)}$$

Similarly, the influence of other weights in the local template can be calculated. To compute $\frac{\partial \vec{w}}{\partial \vec{p}}$, reference is made to Equation (2) to see how the potential can be expressed in terms of the parameters. On solving for Equation (2), weights can be expressed as a function of the parameters. The last term from Equation (7) can be computed as.

$$\frac{\partial \vec{p}}{\partial \vec{t}} = \frac{1}{\partial \vec{t}} (\frac{1}{1 + exp^{-\vec{t}}}) = \vec{p} - (\vec{p})^2$$

4) Weight Update: After each iteration of the policy optimization process, weights are updated to reflect changes in the gradient of the new potential field. Since these weights are represented in a parametric form they are updated by first updating their representing parameters. These parameters, modeled as sigmoid functions, are updated as follows

$$p = \sigma(t) = \frac{1}{1 + exp(-t)}$$

where the sigmoid parameters were updated by gradient descent on the error function using learning rate α.

$$t \leftarrow t + \alpha \vec{\nabla} e$$

The weights for a node, expressed as a function of the parameters, are now updated with the newly computed parameters.

V. EXPERIMENTS AND RESULTS

In the experiments performed here, the environment is represented using a grid map of size 32×32 composed of evenly-spaced cells . This map is a discretized C-space representation of a local environment and is enclosed within an artificially created boundary in order to relax the potential. In this C-space the agent is represented as a point and all obstacles and goals are mapped to grid cells. The harmonic function path planner is used to generate the control policy using Dirichlet boundary conditions. According to Dirichlet's condition the obstacles are set at a fixed maximum potential. Hence every cell that represents an obstacle is given a potential value of 1 and the goal a potential of 0. The cells representing free space are initialized to a value of 0. Motion

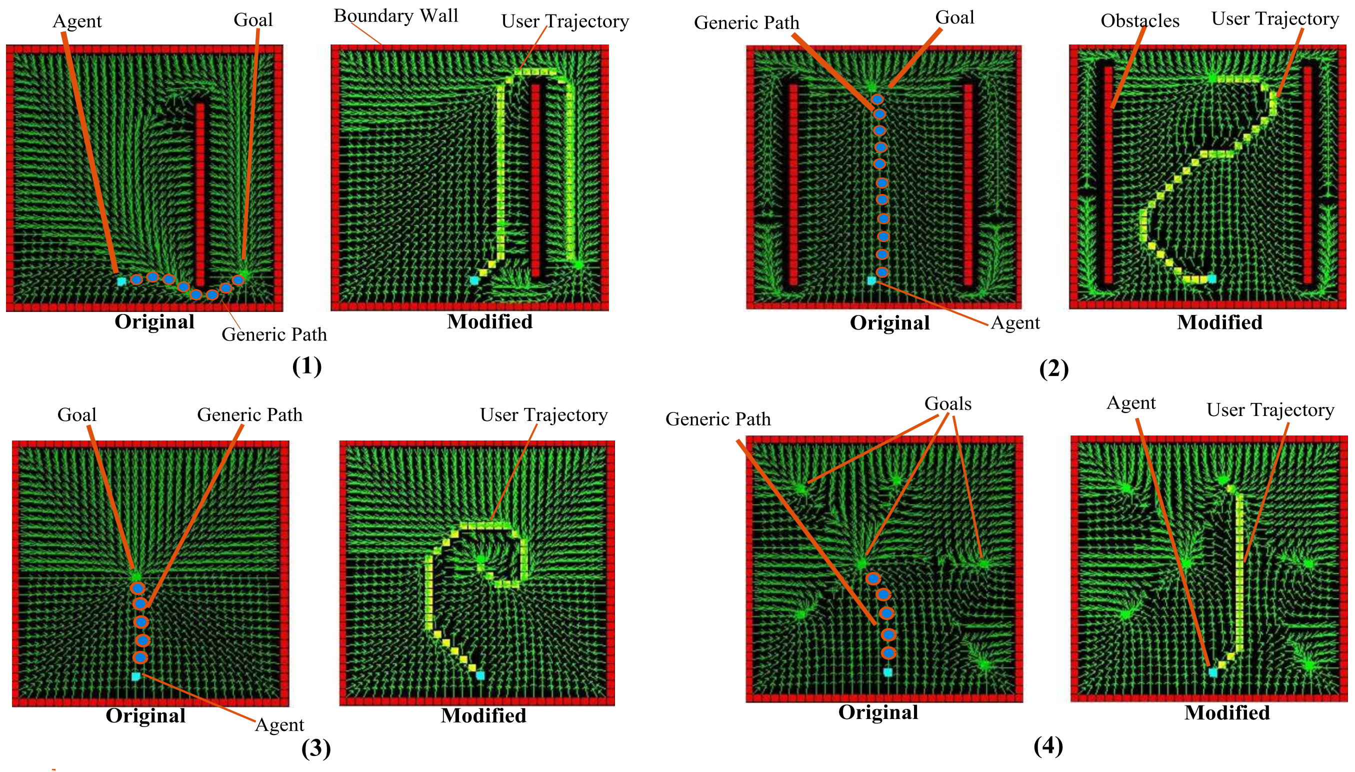

Fig. 5. Original and modified harmonic potentials for Experiment 1 (top left), 2 (top right), 3 (bottom left), and 4 (bottom right)

preferences of a user in a given environment are captured by selecting a path on the grid leading to the goal. The grid is then relaxed using symmetric weights over the entire region to create a potential field. The negative gradient of this potential field is the default generic path that the user would take to reach its goal. The harmonic function path planner then modifies these symmetric weights in order to generate a negative gradient whose gradient direction approximates the user's motion direction which was selected previously.

To investigate the applicability of the approach to arbitrary target trajectories, a series of experiments was performed where user trajectories were hand created. These experiments all required significant changes in the gradient direction.

For all the experiments simple user trajectories were taken to capture motion preferences. Using the trajectories, the weights were then modified to adjust the harmonic potential to reflext the corresponding preferences. Fig.5 shows the original and the customized policies for four of the experiments performed where the dark blocks indicate obstacles and the vector in each cell represents the gradient direction of the harmonic potential. The wall experiment (Experiment 1) shows how a user preference, inidicated by the sequence of cells representing the trajectory, forces the agent to move around the wall positioned to its right. Fig.5(1) shows how the original gradient computed using symmetric weights results in a generic path which passes below the obstacle while the modified path moves around the top of the wall.

The original generic path for an agent moving down a hallway is shown in the left of Fig.5(2). The agent follows the negative gradient down the center of the hallway. If so desired, however, the weights can be trained to lead to a path that causes the agent to move in a S-curve as seen in the right of Fig.5(2). In this experiment it took 104136 iterations until the error fell below 10^{-8}. The learning curve for the weight adjustment is shown in Fig.6.

Fig.5(3) shows a more complex example of a user trajectory modification where the agent is forced in a circle about the goal before reaching it. This is an extremely complex path given that the goal always exhibits attractive force causing the negative gradient to point towards the goal. It took the policy optimizer 268224 iterations to adjust the weights and generate a policy which closely matches the user trajectory. The original generic path computed for this experiment can be see in the left of Fig.5(3). The directional error between the user trajectory and the computed gradient after 268224 iterations can be see in Table I. The learning curve for this experiment can be seen in the middle of Fig.6.

Fig.5(4) shows an example with five potential goals where the gradient computed using symmetric weights causes the agent to select the goal in the center. To introduce a preference for a different goal, a user trajectory is generated which leads to the goal at the top. The right of Fig.5(4) shows the modified gradient computed after extracting motion preference from the user path and as a result, leading to the execution of a different task. It took the policy optimizer 224262 iterations to generate a policy which matches the user selected trajectory. The learning curve for this experiment can be seen on the right in Fig.6.

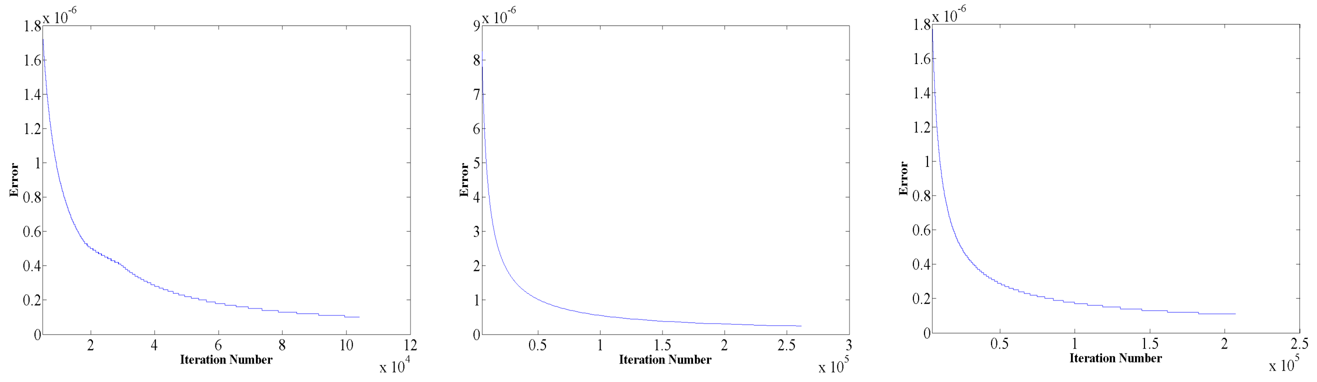

Fig. 6. Learning curves for Experiments 2, 3, and 4 (from left to right)

TABLE I
ANGLE ERRORS FOR EXPERIMENT 3

Grid Loc.	Angle Diff.(Radians)	Grid Loc.	Angle Diff.(Radians)
[14][21]	0.06848875	[15][21]	0.05435131
[16][21]	0.04183170	[17][21]	0.04436677
[18][21]	0.07271755	[19][21]	0.10047857
[13][20]	0.08915985	[20][20]	0.11065160
[12][19]	0.08554078	[21][19]	0.08803131
[11][18]	0.08507000	[21][18]	0.08777382
[10][17]	0.06842601	[21][17]	0.05749669
[10][16]	0.17441738	[16][16]	0.06088347
[21][16]	0.16680097	[10][15]	0.04815570
[17][15]	0.03193391	[21][15]	0.00694400
[10][14]	0.13430074	[18][14]	0.03863792
[19][14]	0.05785567	[20][14]	0.07075152
[10][13]	0.05264248	[10][12]	0.12093754
[10][11]	0.05311862	[10][10]	0.07762738
[10][9]	0.02983270	[11][8]	0.07026553
[12][7]	0.03069032	[13][6]	0.06258808
[14][5]	0.02431623	[15][4]	0.06838232

VI. CONCLUSIONS AND FUTURE WORK

To capture user motion preferences and allow a robot or artificial agent to generate corresponding trajectories, this paper presents an approach to customized path planning where a harmonic function path planner is modified by changing its underlying parametric representation. Harmonic function parameter modification here requires user trajectories as a reference for desired paths and uses them to compute an error function which expresses the angular difference between the user paths and the trajectories generated by following the gradient of the harmonic potential. Using a local assumption about the influence of parameters on the directional error, and employing error propagation techniques, gradient descent in parameter space is performed on this error function in order to allow the path planner to generate paths that approximate the user's reference trajectories. The policy is here iteratively modified according to the gradient until the desired user trajectory is approximated. Various complex user paths were considered to evaluate the performance of the policy optimization algorithm and results observed have shown closely matching paths produced by the harmonic function path planner. Using this, the planner can be used to achieve more customized trajectories for the control and motion planning of semi-autonomous wheelchairs, semi-autonomous mobile robots, robotic arms or intelligent game characters.

A. Future Work

Some paths could be extremely complex for the harmonic function path planner to learn through weight modification. To alleviate this, additional methods for the safe modification of harmonic potentials, including the use of constrained potential field relaxation at selected locations will be studied. Another issue is that not every motion preference can be captured from user trajectories alone. It might thus be necessary to also learn personalities, behaviors, conditions and use them to infer hidden preference patterns.

REFERENCES

[1] J. A. Coelho, R. Sitaraman, and R. A. Grupen, "Parallel optimization of motion controllers via policy iteration," in *Advances in Neural Information Processing Systems 8*, Denver, CO, 1995, pp. 996–1002.
[2] G. Arechavaleta, J.-P. Laumond, H. Hicheur, and A. Berthoz, "Optimizing principles underlying the shape of trajectories in goal oriented locomotion for humans," in *Int. Conf. Humanoid Robots*, 2006.
[3] D. C. Brogan and N. L. Johnson, "Realistic human walking paths," in *Int. Conf. Comp. Animation and Social Agents*, 2003.
[4] C. I. Connolly and R. A. Grupen, "On the applications of harmonic functions to robotics," *J. Robotic Systems*, vol. 10, pp. 931–946, 1993.
[5] S. P. Singh, A. G. Barto, R. Grupen, Christopher, and C. Connolly, "Robust reinforcement learning in motion planning," in *Advances in Neural Information Processing Systems 6*, 1994, pp. 655–662.
[6] M. Kazemi, M. Mehrandezh, and K. K. Gupta, "Sensor-based robot path planning using harmonic function-based probabilistic roadmaps," in *IEEE Int. Conf. Robotics Automat.*, Seattle, WA, Jul 2005, pp. 84–89.
[7] M. Kazemi and M. Mehrandezh, "Robotic navigation using harmonic function-based probabilistic roadmaps," in *IEEE Int. Conf. Robotics Automat.*, New Orleans, LA, Apr 2004, pp. 4765–4770.
[8] M. Trevisan, M. A. Idiart, E. Prestes, and P. M. Engel, "Exploratory navigation based on dynamical boundary value problems," vol. 45, no. 2, pp. 101–114, 2006.
[9] F. Dapper, E. Pretes, M. A. Idiart, and L. P. Nedel, "Simulating pedestrian behavior with potential fields," in *Advances in Computer Graphics*, vol. 4035, Sep 2006, pp. 324–335.