ADAPTING HARMONIC FUNCTION PATH PLANNING: TO REFLECT USER MOTION PREFERENCES

by

GILES JOHN D'SILVA

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2008

To my mother father and sister

ACKNOWLEDGEMENTS

Thanks to my God for everything he has done for me. I am enormously thankful to my parents for the encouragement and financial support they have provided and for believing in my abilities to succeed. It would not have been possible to pursue my masters degree if not for my litter sister who took care of my mother while I was away and for always cheering me up when I needed the most.

My uttermost thanks to my advisor Dr. Manfred Huber, for being so patient and supportive in my research work. I am grateful to him for devoting hours together in order to make sure I understand the concepts well. It was also a pleasure talking to him and being advised in areas right from technology to soccer. Thanks to Dr. Zaruba and Mr. Levine for being on my committee and devoting their time to attend my defense. I would also like to thank the professors from alma mater Chowgule College, Mrs. Sameena, Dr. Adhikhari, Mr. Kumaresh, Mrs. Sangeeta and to all the professors who motivated and guiding me along the way. Special thanks to my development manager at Siemens Mr. Nagendra for urging me to pursue a master degree in the US.

There were a lot of friends who helped me with my research work along the way. Thanks to John, Shawn and Savio for critiquing and reviewing my thesis work, to Sahil for the company in video game play that kept me visiting the lab ever so often and to my lab mates Vamsi for keeping a check on the progress of my thesis writing and for accompanying me during late nights research work, to Big John for providing his books for reference and for the many games of Wii and Xbox, to Vidhya for constantly lecturing me on my lazy irresponsible attitude towards my research work

and to Dr. Darin for the helpful tips on research and for introducing me to writing my thesis in latex. To all those who have helped me in my thesis, whose names have not been mentioned I thank you from the bottom of my heart.

Lastly, my appreciation towards The University of Texas at Arlingtons Computer Science department for all the financial aid and the education received.

November 25, 2008

ABSTRACT

ADAPTING HARMONIC FUNCTION PATH PLANNING:
TO REFLECT USER MOTION PREFERENCES

Giles John D'Silva, M.S.

The University of Texas at Arlington, 2008

Supervising Professor: Manfred Huber

Every human in the real world has a unique motion preference while moving to achieve a given task. These preferences could be expressed by moving in a straight line, following the wall along a corridor, avoiding sharp turns, preferring hard flat surfaces over damp uneven surfaces, choosing the shortest path to the goal or by giving a high priority towards safety by maintaining a definite distance from obstacles. To automatically incorporate user preferences into motion planning we could extract the trajectories taken by the user to achieve a task in a given local environment and attempt to use them in potentially similar environments. However these trajectories can not be easily generalized and the intermediate paths can generally not be inferred by interpolation between similar paths since the resulting path may not always lead to the goal or could sometimes even collide with an obstacle. As a result a motion planner using sample trajectories and path interpolation would lose the characteristic of a good path planner and may also fail to correctly represent the user's motion preferences while transferring such trajectories into more dissimilar environments. Motivated by this, the goal of this research is to design a path planner that

is able to transfer user motion preferences in a parametric form to new similar local environments and to generate paths that are smooth, complete and correct. In this research we modify a harmonic function path planner to model the preferences of a user's motion as parameters which could then be used to generate new paths based on these preferences for potentially similar environment configurations. To model such preferences the representing parameters have to be learned using a machine learning algorithm. The algorithm extracts the trajectory data for the user whose preference we are trying to capture and computes the desired path (the harmonic function gradient) direction from every trajectory point. Next, it initializes the parameters to default values that generate a generic path to the goal. We then modify and update these parameters until a path is generated that matches the desired direction followed by the user. These parameters now represent the user's motion preference in that local environment and can be transferred into new, similar environments. We then input these parameters into our path planner to generate a path for this new environment. Since the path is generated by a harmonic function path planner it is complete and has no local minima or maxima and the user is assured of reaching the goal if a path exists. This customization of the path planner to learn user motion preferences could have potential applications in autonomous vehicles, semi autonomous wheel chairs, remote control of robotic systems or the creation of custom game characters.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Autonomous agents in the real world are normally assigned navigation tasks which require them to rotate or translate in order to achieve their goal. Motion autonomy is thereby frequently addressed by a planner which is responsible for generating a plan consisting of a sequence of actions needed by an agent to achieve its goals. For a planner to propose a plan, it first needs information about how the environment is configured, which is usually provided by the agent by sensing the local environment and feeding the information into the planner. The planner then uses the defined motion planning algorithm to create a policy for the agent. The policy could represent a direction vector from every state in order to reach its goal. Paths generated by a path planner should be complete, legal and should maintain safe distances from obstacles. However these are not the only criterias that make a path desirable for motion. In robot path planners the paths are usually computed according to simple metrics. On the other hand human motions to achieve a similar task are usually distinctive and particular to their individualistic preferences. For example, routes taken by people driving from the same neighborhood to reach a common factory unit usually differ. Some take the shortest possible route that can bring them to work early; others prefer taking quieter scenic inroads, avoiding highways, while the remaining could have preferences for the terrain. To personalize the autonomous generation of paths, the planner could be pre-programmed to generate a path 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 motion objectives in new environments. The question we are trying to address in this thesis is how a generic path planner can be modified to create personalized paths for an agent in such a way that these motion preferences could be transferred between potentially similar environments. This research is an extension of previous work done in modifying harmonic function controllers [1] [2] with the goal to allow the path planner, to generate customized paths. The path planner chosen for the implementation is a harmonic function path planner due to its flexibility and completeness and correctness properties. The harmonic function planner was selected for this research work because of its robustness in the presence of unanticipated obstacles and errors, completeness, ability to exhibit different useful modes of behavior, and rapid computation [3]. The constraints or (Boundary Conditions) used by the harmonic function motion planner assure that the agent maintains safe distances from obstacles. Work done in [1] and [2] demonstrates that harmonic functions, when expressed in a parametric form, can be useful in modifying the generated path by adjusting its parameters. The framework of our research is to analyze the motion of an agent in the given configuration space and to learn preferences from user specified trajectories and transfer them to similar environment configurations. The assumption of having similar environments is made as the problem of finding matching environments has not been addressed in this thesis. Harmonic functions are solutions to Laplace's equation which generate potential fields with no local minima or maxima. Harmonic function potential field values in their generic form represent the probabilities of hitting an obstacle assigned to the nodes in a discretized environment prior to arriving at the goal while performing a biased random walk [4] [5]. This potential at each node is expressed as a weighted average over its neighboring potential values, thus assuring the presence of

an ascending and a descending gradient on this field when generating a policy for the agent. Motion preferences are captured in the form of weight modifications that are then used in the path planning process to generate paths with similar characteristics. These motion preferences are assumed to be represented through a set of trajectories. The direction vectors constituting the trajectories are the user specific information about preferences that the planner utilizes to optimize its policy. In Chapter 3, a detailed description is presented on how weights are modeled and modified to minimize the directional error between the user trajectories and the computed gradient. Since the influence of different weights on the error decreases exponentially with distance, a local assumption of weight influence is also discussed in this chapter. Chapter 3 also explains how the error from the path nodes can be propagated through an overlaid feed forward network. Chapter 4 shows the implementation of the experiments and the observation made on the results. Finally, conclusions were made in Chapter 5 on the results which were achieved using the implementation described in this thesis work.

CHAPTER 2

RELATED WORK

2.1 Functionality of a Path Planner

2.1.1 State

In motion planning, the state of an agent could represent the location and orientation of the agent in a given configuration of the environment. Knowledge of the entire state space of the agent which could arise is not absolutely necessary for planning. Given the current state of the agent, the path planner suggests the necessary action to be taken. Some motion planners do not require the initial state of the agent to compute a policy. A harmonic function path planner used in this research is an example of a motion planner not requiring an initial state for an agent to generate a motion plan. This is achieved by computing a negative gradient over the entire local environment.

2.1.2 Actions

Actions are applied to change the state of the agent. The path planner is responsible for generating a sequence of actions required to change between the intial state and the desired state. How a state changes when a given aciton is applied can be expressed as a state-transition function in case of discrete time or as an ordinary differential equation for continuous time. Actions specified by the planner could be a rotation or translation for motion planning.

2.2 Applications of Planning Algorithms

Planning algorithms have widespread applications in various areas of the industry like robotics, manufacturing plants, drug design, medical surgeries, aerospace applications, warfare and video games.

Planning algorithms have been able to solve board puzzles like chess, sudoku and rubik's cube. Discrete planning which is applied to solve these types of problems is also used in planning the navigation of robots in a 2D grid.

The motion planning software developed at Kineo CAM is used in the automotive assembly task to insert and remove a windshield wiper motor from a car body cavity [6]. Many such automotive tasks are solved by planning the motion of robotic arms to assist on the assembly line. Motion planning software developed by the Fraunhofer Chalmers Center is used by the Volvo Cars (in Torslanda,Sweeden) assembly plant for the sealing process of their car bodies using programmed robots to function automatically [6].

Motion planning also has its application in video games. The motion of intelligent game characters in video game can be automated by specifying the behavior of each character from a very high level of abstraction. Humanoid robots use motion planning algorithms to achieve near human like characteristics. The latest Asimo robot from Honda was designed and programmed to walk at 3km/hr [6]. Kineo CAM also developed software for nonholonomic path planning which was designed to transport portions of the Airbus A380 across France [6]. The software planned routes through villages avoiding any collision with obstacles along the path and maintaining the differential constraints imposed by 20 steering axles [6].

Path planning algorithms could also be used to derive trajectories for vehicles moving at very high speeds [6]. This involves dynamic constraints, uncertainties and obstacle avoidance. Planning algorithms have also been used in computational biology

to solve the docking problem which requires determining if flexible molecules can insert themselves into a protein cavity [7]. Probabilistic Roadmap motion planning techniques (Used in robot motion planning) have also proven successful at studying protein folding pathways and potential landscapes. [6]

2.3 Modelling the Environment

In order for the agent to navigate in a well defined local environment it has to create a map of the environment. This map needs to be updated depending upon a change in surroundings of the local region. The map generated is important so that the path planner can compute a policy for the agent in this environment.

An office environment could be considered for navigation in which the agent is assigned a motion task. To generate a map of the environment the agent can use various sensors to determine the geometry of the environment but these sensors can also impose a number of practical limitations while sensing the environment, for example, through limited sensor range, sensor noise, odometric error due to slippage or drift, complexity of the environment, etc. These limitations influence the decision of choosing the right kind of map. There are 2 main paradigms in constructing maps for an agent's environment, namely metric and topological maps. Grid-based and feature maps are subsets of metric maps. These maps are discussed further in terms of how they are created for local environments and their advantages and disadvantages are also presented.

2.3.1 Grid-Based Maps

Grid-based maps as a subset of metric maps, divide the envrionment into grid cell. Each grid cells contains information about the envrionment and could be marked as an obstacle, free space or a goal. Grid based maps are relatively easy to construct

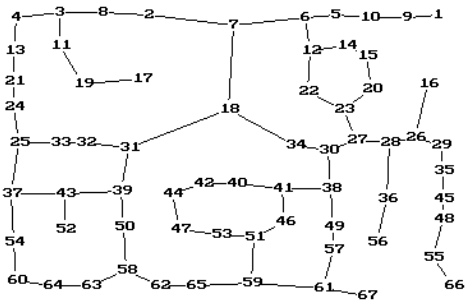

Figure 2.1. Topological map [8].

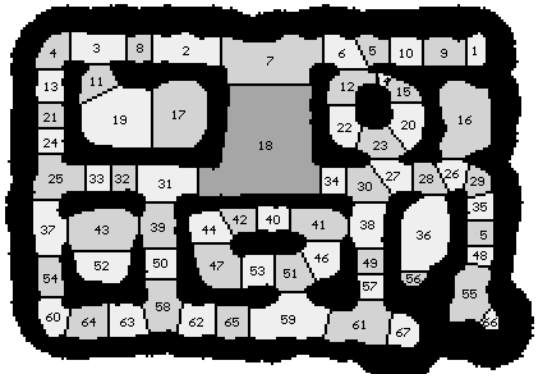

Figure 2.2. Topological region with critical lines [8].

and maintain as their resolution is independent of the complexity of the environment. However, their time and space complexity increases for large environments. Corrections for slippage or drift are absolutely necessary when the agent navigates while sensing the environment. The error caused due to drift are eliminated using internal encoder information and the agent is localized using correlation between the local and global maps and by memorizing global wall orientation. The implementation of our research work uses grid based maps to represent the local environment and the agent.

2.3.2 Feature Maps

Sonar or laser sensors can be typically used in sensing the environment in order to build feature based maps. The typical features in a feature based map for a local environment are the agent, straight walls and polyhedral objects. The agent continuously observes these features for local referencing. SLAM techniques along with Kalman Filters and particle filters can be used to create feature based maps. These methods provide localization of the agent at virtually any point in the local environment. Their complexity grows with larger environments and number of map objects.

Some of the issues with creating feature maps arise due to limited sensor range, limited field of view, occlusions and noisy data [9]. Probabilistic frameworks for localization have been proposed to overcome these issues.

One of the experiments carried out in [9] shows the feature based map as seen in Figure.2.3 of the Belgioioso Castle, an exhibition site in Italy. A manually controlled Pioneer robot (from the University of Freiburg) explored this environment with a trajectory of 228m in 16'27". The 227 features were mapped and two loops of approximately 100 m were correctly closed in this experiment. It can be seen that this map also closely resembles an architectural blueprint of the environment.

2.3.3 Topological Maps

These are simplified maps that represent the environment using a graph based approach. Nodes in the graph represent important landmarks like doors, entry and exit points etc and arcs represent's that a non obstructed direct path exists between them. Topological maps are built by partitioning regions with critical lines joining critical points. Critical lines can be considered to be doorways and hallways and critical points can be the exit and entry points in the map. There are a few drawbacks

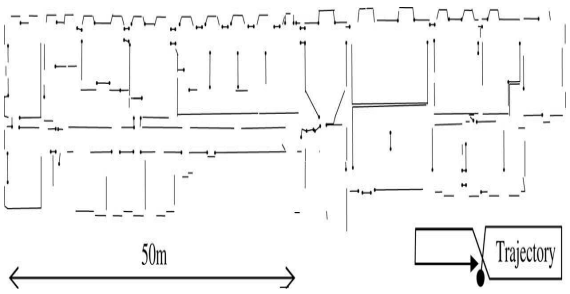

Figure 2.3. Feature Map [9].

of using topological maps as they suffer from incorrect place recognition in situations where places look alike or if the same place has been sensed from different viewpoints by taking different paths. Time complexity for constructing and maintaining topological maps for complex environments increases due to the increase in the number of critical lines or arcs.[8]

2.4 Configuration Space

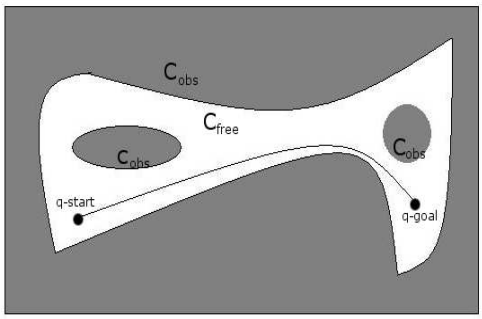

Figure 2.4. Configuration Space.

Configuration spaces have been proposed in the popularized work by Perés [10]. The Configuration space is also known as the manifold of the agent and often represented as C-space. Configuration spaces can be referred to as a representation tool for the motion planning of an agent. C-spaces provide an abstraction in the representation of a complex dimensional agent as it transforms the agent into a single point in this space. This simplifies the motion planning of the agent is this space. The dimensions of a manifold (C-space) are equal to the degrees of freedom of the agent. Obstacles are also mapped into this same configuration space. Let C be the configuration space, C_{free} the space in which the agent can move freely without colliding, C_{obs} the space occupied by obstacles in configuration space then C can be represented as

$$C = C_{free} \cup C_{obs}$$

where $C_{obs} = \cup_{i=1}^{q} C_{obs_i}$, and q is the number of obstacles in this space.

2.5 Harmonic Function Path Planner

The process of motion path planning consists of at least 2 stages, the planning stage and an execution stage. In the planning stage the path planner computes the desired policy so that the agent can reach its goal from the given initial start position. The planning algorithm defined is responsible for the computation of this policy. The execution stage consists of how the agent actually moves along the path, differential constraints and dynamic constraints are taken into account during path execution. The harmonic function path planner uses the potential field approach for path planning. This approach has proven robust for path planning in real time environments as described in the next sections.

2.5.1 Potential fields

Potential fields have been originally proposed by Khatib for collision avoidance when the map of the environment is computed in real-time by sensing during path finding [11]. According to Khatib an agent in configuration space is acted upon by imaginary forces. Goals produce attractive forces and obstacles produce repulsive forces pushing the agent away and then preventing collisions. The path taken by the agent is a resultant vector of these artificial forces acting upon it in a given free space. Since frequently importance was given to real-time planning over actually finding the goal, major limitations of many potential fields are local minima that get created in artificial potential fields, causing the driving force to vanish and the agent to get trapped before it reaches its goal. To escape from a local minimum vector field histograms (VFH) were proposed [5] [12] [4].

The field of artificial forces $\vec{F}(q)$ in C (configuration space) are produced by a differentiable potential function $U : C_{free} \rightarrow \mathbb{R}$,with: $\vec{F}(q) = -\vec{\nabla}U(q)$, where $\vec{\nabla}U(q)$ denotes the gradient vector of U at q. We take the negative gradient because we are performing a gradient descent on the potential field in which obstacles are set at a maximum and goals to a minimum.

$$U(q) = U_{att}(q) + U_{rep}(q)$$

and

$$\vec{F} = \vec{F_{att}} + \vec{F_{rep}}$$

Where,

$$\vec{F_{att}} = -\vec{\nabla}U_{att}$$

and $\vec{F_{rep}} = -\vec{\nabla}U_{rep}$ are the attractive and repulsive forces respectively.

2.5.2 Laplace's Equation

The scalar form of Laplace's equation can be represented as

$$\nabla^2 \phi = 0$$

where ∇^2 is the Laplacian operator. If a function ϕ satisfies Laplace's equation then it is called a harmonic function.

A harmonic function ϕ of two variables, x and y, which satisfies Laplace's equation can be expressed as

$$\nabla^2 \phi(x, y) = \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2}$$

2.5.3 Harmonic functions

A function ϕ defined on a domain $\Omega \subset \Re^n$ is said to be harmonic if it satisfies Laplace's equation:

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

Now there is more than one solution that satisfie's Laplace's equation depending on their boundary conditions. However, every solution is free of local minima and other critical points except for saddle points which may exist. An exit from these points can be found by searching in their neighborhood. The gradient vector field of a harmonic function has zero curl and so a gradient descent on this vector field always directs an agent towards the goal from any point in this field. The streamline generated from the trajectory points is smooth for any point along the trajectory [13] [6].

2.5.4 Characteristics of Harmonic Functions

Harmonic functions under the specified boundary conditions generate legal paths from any point in the configuration space if they exist and hence the path generated is said to be complete. The path taken is a gradient descent of the poten-

tial field. The potential field has no local maxima or minima because the potential at a point is computed by taking an average of its neighbors. Hence, the only types of critical points which can occur are saddle points. Saddle points are stationary points but not extrema; to escape from a saddle point an exploration in the neighborhood for a negative gradient places the agent back on the path to finding the goal. Saddle points can only occur in infinite precision dynamics and thus do not cause any major problems in real world robotics. Using a harmonic function, the path planner can be modeled as a local reactive planner where the environment details are updated incrementally and are known only in real time, this makes it extremely robust and flexible [3] [14] [15] [16]. . From experimentations performed in [16] it is seen that harmonic functions work well in dense environments. A dense environment according to [15] is one in which obstacles are in the range of the sensors which are mounted on the agent.

2.5.5 Numerical Solutions to Laplace's Equation

There are various methods used for solving a system of linear equations. Since we are finding solutions to Laplace's equation we have a homogeneous system. Direct or indirect techniques can be used in finding the solutions to these equations. The Gauss elimination technique can be considered a direct method as it uses the triangular form approach in which it tries to eliminate some variables from the equation by first solving for them. This is however computationally expensive. Another approach considered is the indirect or approximation techniques that are used to solve the system of linear equations.

In Gauss-Seidel iteration an approximate value is chosen for the unknown variables where in this case are initially approximated to zero as the boundary condition to relax the potential field equals to one and the lowest possible potential assigned

to a node or cell in this field is 0 (goals). Let the potential value at a given node be calculated as an average over its neighbors. To relax the potential the entire field is scanned and updated repeatedly until the change in potential values falls below a residual value. Gauss-Seidel is different from Jacobi iteration as it uses the newly computed values for solving the rest of the potential values.

$$\begin{aligned}\phi^{(k+1)}(x_i, y_j) = \frac{1}{4}(&\phi^{(k+1)}(x_{i-1}, y_j) * w_1(x_i, y_j) \\ &+\phi^{(k)}(x_{i+1}, y_j) * w_2(x_i, y_j) \\ &+\phi^{(k+1)}(x_i, y_{j+1}) * w_3(x_i, y_j) \\ &+\phi^{(k)}(x_i, y_{j-1}) * w_4(x_i, y_j))\end{aligned}$$

As we can see from the equation above we use the newly computed potential values of our left and top neighbors. This half of the neighbors are from iteration $k + 1$, while the other half are from iteration k. Jacobi iteration can be represented as

$$\begin{aligned}\phi^{(k+1)}(x_i, y_j) = \frac{1}{4}(&\phi^{(k)}(x_{i-1}, y_j) * w_1(x_i, y_j) \\ &+\phi^{(k)}(x_{i+1}, y_j) * w_2(x_i, y_j) \\ &+\phi^{(k)}(x_i, y_{j+1}) * w_3(x_i, y_j) \\ &+\phi^{(k)}(x_i, y_{j-1}) * w_4(x_i, y_j))\end{aligned}$$

This approximation techniques requires a higher computation time compared to Gauss-Seidel iteration as it does not use the newly computed values of its previous nodes in the current iteration as seen in the equation above [17].

2.5.6 Bondary Conditions

Solutions to Laplace's equation can be computed using two restricted forms of boundary conditions, namely Dirichlet and Neumann In the Dirichlet boundary condition the potential at the boundary is fixed to a constant maximum value in the

configuration space. All obstacles within the configuration space are also represented by a constant maximum value so that the potential flow is an outward normal to the obstacles surface. In Neumann boundary conditions the gradient vectors are forced to be tangential to the obstacle boundary surface causing the agent to graze along the obstacle boundary which may not be preferred for the motion of some agents. A harmonic function can be computed by taking a linear combination of the two boundary conditions mentioned above resulting in the agent moving at an angle along obstacle boundary and maintaining a safe distance. $\phi = \lambda\phi_{Dirichlet} + (1-\lambda)\phi_{Neumann}$ where $\lambda\epsilon[0,1]$. $\lambda \leq 1$ avoids shallow gradients sometimes found in Dirichlet's solution. The resulting ϕ is harmonic and obeys the min-max principle thus generating paths that are complete and correct [3] [18].

2.6 Manhattan Distance

There are two main types of metrics used to measure distance, the Eucledian distance and the Manhattan distance.

Euclidean distance between 2 points (x_1, y_1) and (x_2, y_2) in 2D Euclidean space is defined as

$$\sqrt{(y_1 - y_2)^2 + (x_1 - x_2)^2}$$

Manhattan distance between 2 nodes positioned at (x_1, y_1) and (x_2, y_2) in a descretized space can be defined as

$$|(y_1 - y_2)| + |(x_1 - x_2)|$$

The distance between cells or nodes in a 2D grid can be computed recursively for each cell. All neighbors to the target cell are assigned a label of 1. Then, neighbors to each of these neighboring cells are recursively labeled with the distance increasing by

1 at every recursive step. The labeling continues until all required cells are assigned a distance to the target cell.

Manhattan distance is used in the implementation of our experiments to label nodes of the network in order to propagate moion preferences to nodes that are not part of a defined trajectory. These nodes act like hidden nodes of feed forward neural network and create internal layers in the network.

2.7 Feed Forward Neural Network

2.7.1 Neural Network

Neural networks (NN) had been first proposed in 1943 as simple mathematical model of a neuron. An artificial neural network is an abstract, simplified model of how neurons of a biological nervous system process information. These models do not process information sequentially; they have a hierarchical multilayered structure so that information can be transferred to distant units in parallel. Neural networks have been used to solve many research problems in the field of computation including problems in classification, function approximation and data processing. These networks are composed of nodes which are interconnected with directed links and connector nodes that could be labeled as an input, output or hidden node. Hidden nodes are nodes which lie in the internal layer of a network and thus have no direct connection to the input or the output of the network. Numeric weights are assigned to each link of the network to determine the connectivity strength. These bias weights modify the output generated at a node. The output value generated varies depending on different functions which are used to compute the output of a node's value depending on the information being processed.

Consider a model of a neuron with n inputs with each input being connected by a link which has a weight attached to it. The transmitted information or output is a sum of products over all input values and the associated weight. The activation function is used to introduce non-linearity in the network. The output value at this node is then evaluated. An artificial neural network is a collection of such nodes. Let $x_i, i \in [1..n]$ be the set of inputs and $w_i, i \in [1..n]$ be the set of associated weights, then the output at a node can be represented as

$$output = g(f(x, w)) = g(\sum_{i=1}^{n}(x_i * w_i) + t)$$

where t is the bias term.

Figure 2.5 shows the representation of a simple node in the neural network.Some of the popular models of neural networks used are the feed-forward neural network (FFN), radial basis function (RBF) network and the Hopfield network. The next section explains how the feed-forward network is designed.

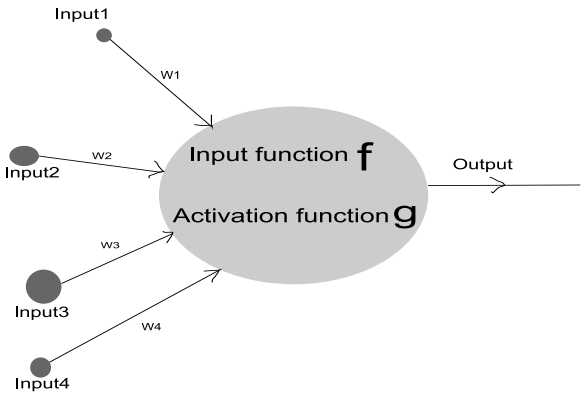

Figure 2.5. Model of a Neuron.

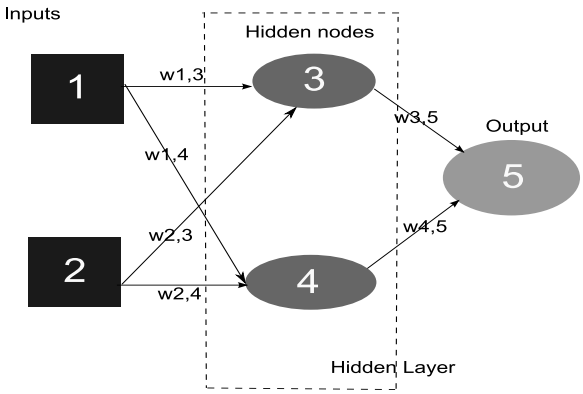

Figure 2.6. Model of a Feed Forward Network.

2.7.2 Feed-Forward Network

Feed forward networks organize the nodes into layers where nodes in a layer propagate information to other layers in a forward sequential manner to avoid loops. The output of every node is expressed as a function of its input in this type of network which can be single or multilayered. A multi layered feed-forward network is shown in Figure 2.6 This simple network has two input nodes 1 and 2 that feed information into the network. Nodes 3 and 4 are hidden nodes in this layer of the network and node 5 is an output node. $W_{i,j}$ represent weights which act as a bias to achieve a desired output value. These weights are later useful in training the network to adjust the computation in order to minimize the error of the desired output value. Since each node is a function of its input nodes. The output value at node 5 is computed in the following equation

$$Output(node5) = g(W_{3,5}I_3 + W_{4,5}I_4 + t_5)$$

$$= g(W_{3,5}g(W_{1,3}I_1 + W_{2,3}I_2 + t_3) + W_{4,5}(W_{1,4}I_1 + W_{2,4}I_2 + t_4))$$

Where I_i is the input at node i

2.7.3 Error Backpropagation

In a backpropagating neural network the error gradient is propagated backwards in a feed forward artificial neural network. Error propagation is performed iteratively to train the multilayer feed forward artificial neural network in order to minimize the given error function which could be a sum-of-squares error function. In order to generate the required output in a feed forward neural network given a set of input values the network has to learn which set of weight values would generate the desired output. These weights can be adjusted by human supervision for a simple network but may be extremely difficult for multi-layered networks that contain a large number of hidden nodes. Feed forward networks are supervised networks since they require a desired final output to be provided. The network learns from feedback in order to produce the desired output. This is achieved through weight modifications. The error at the output node is first evaluated and the weights connecting the last hidden layer to the output node are adjusted to generate the desired output. These weight updates are then propagated backwards to the weights connecting it to the previous hidden layer. This process of transferring the error in the backward directions is termed error backpropagation. This procedure is similar to feed back control system where the amount of change required to minimize the error is looped back into the system. The error function that is normally minimized is a sum of squared error. A learning rate is assigned in training the network which specifies by how much the weights need to be updated. A higher learning rate may accelerate the error minimization but may overshoot the desired output in some occasions [19][20]. Let e be the error function to minimized and $O_{desired}$ be the desired output and O the current output value, then the error is usually represented as

$$e = \sum \frac{1}{2}(err)^2 = \sum \frac{1}{2}(O_{desired} - O)^2$$

The output error can be reduced by performing a gradient descent on the squared error by calculating the partial derivates of e with respect to each weight W_j

$$\begin{aligned}\frac{\partial e}{\partial W_j} &= err\frac{\partial err}{\partial W_j} \\ &= err\frac{\partial}{\partial W_j}(O_{desired} - O) \\ &= err\frac{\partial}{\partial W_j}(O_{desired} - \sum W_j.I_j) \\ &= -err\Delta_j\end{aligned}$$

where I_j are the input values at node j and W_j are the weights from node j. Then each weight is updated as follows:

$$W_j \leftarrow W_j + \alpha err\Delta_j$$

where α is the learning rate.

2.8 Different Path Planner Optimizations to Harmonic Functions

Harmonic functions for path planning were first proposed by Connolly and Grupen in 1993 [3] and have since been used in the path planning of mobile robots. Harmonic functions haven been implemented under different classes of boundary conditions, including Dirichlet and Neumann boundary conditions or the combination of the two to exhibit different gradient patterns. The control policy generated by a harmonic function path planner is generic in nature and paths that are stereotyped by user preferences remain unexplored. Research work done by Coelho [1] and Fabio [2] have shown how the planner can be modified to produce alternating paths.

2.8.1 Combination of Different Boundary Conditions

Paths generated by the harmonic function path planner can be altered by using various boundary conditions. The boundary conditions used can give rise to dif-

ferent harmonic functions and the paths which are proposed vary with respect to safety. Dirichlet and Neumann boundary conditions result in two harmonic functions $\phi_{Dirichlet}$ and $\phi_{Neumann}$. The paths generated from $\phi_{Dirichlet}$ cause the agent to move perpendicular to obstacles. However, paths computed from $\phi_{Neumann}$ move parallel to obstacle boundaries. In [21] a new harmonic function was formulated by taking a linear combination of the two boundary conditions. This new harmonic function ϕ_{DN} can be represented as.

$$\phi_{DN} = (1 - \lambda)\phi_{Dirichlet} + \lambda(\phi_{Neumann})$$

Where,

$$0 \leq \lambda \leq 1$$

A reinforcement learning algorithm called Q-learning was then used to learn the mixing parameter λ to generate minimum time paths from every point in the environment to the goal. This approach towards path modification has limitations with respect to the paths that can be generated. For example, an agent move along the right side of the hallway under the ϕ_{DN} boundary condition can not be redirected to move along the left wall. Such path modifications can only be achieved using modified asymmetric weights.

2.8.2 Control Policy Optimization

The research work done 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. These conductances are analogous to the symmetric weights that connect the nodes of a grid. The methodology used to update

these conductances is achieved using a policy iteration algorithm. The control policy is repeatedly modified until the optimal policy generates a gradient which optimizes the given performance index. Optimization of the policy was performed over a grid equivalent circuit (Equivalence established using Thevnin's theorem) approximation of the grid to insure a gradient is available for any given initial state in the domain. The paths generated are safe and the capabilities to reach its goal (correctness) are preserved at every step of the policy iteration. Experiments were performed to modify the conductances such that the travel distance to the goal was minimized. This result demonstrated its effectiveness in achieving various desired motion trajectories.

2.8.3 Simulating Pedestrian Behavior

Different solutions to Laplace's equation were previously considered that do not possess local minima. According to Trevisan [15] there are a number of potential functions that do not posses local minima. Fabio [2] proposed adding external force fields, to counteract the natural tendency of the agents that follow the direction provided by the planner. The function suggested creating a potential field that computes an average over its four neighbors and adds an extra term to it. The additional term is used to break the symmetry of the vector field creating a bias towards a desired modified path. The vector field is made asymmetric by the addition of this external force field which is represented by a behavior or bias vectors. These bias vectors capture different agent preferences like the physical limitation, personality, mood and reasoning to produce smooth, safe paths containing no local minima. The equation to compute the potential, when rearranged, expresses a weighted average of its neighbors by maintaining the symmetry between the weight sum across the X and Y dimensions. As a result of this limiting factor, the vector field cannot be completely

customized to produce every desired motion path. In our research these weights have been modified breaking the symmetry to generate preferred paths.

CHAPTER 3

METHODOLOGY

3.1 Harmonic Fuction path planner

A modified harmonic function path planner is used in this research work to generate customized policies. The distinctive paths which the planner generates are a direct resultant of the preferences provided by a user specified trajectory. In this chapter we discuss how the preferences are represented as parameters and how these parameters are learned in order to transfer them to new potentially similar environments to generate smooth, complete paths.

The paths generated by a standard harmonic function planner with a specified boundary condition are always complete and correct provided a legal path exist to the goal. The problem is that these paths are generic in nature and may not be preferred by an agent, even though they adhere to all the characteristics of a good path. These standard path requirements are necessary but are not the only attributes necessary to personalize paths for an agent. In this section we describe the different ways in which a harmonic function planner can be modified to adapt to user motion preferences, such as walking down the right side of a hallway, avoiding sharp turns, preference over terrains, how the goal needs to be approached, moving along the center of a floating bridge, avoid navigation through certain neighborhoods, or the influence that different types of obstacles have on the path as some objects may have a greater force of repulsion on an agent. There could be many such examples that are characteristics resulting in distinctively customized paths. Different techniques have been applied to customize the harmonic function path planner. Firstly, potential

fields generated by harmonic functions can be modified directly by adding constraints on how the potential values in the given domain are relaxed. Other methods include using different functions that are solutions to Laplace's equation [2], or the application of optimization techniques like adjusting the parameters of the current harmonic function [1] to better understand the influence it has on the policy generated. These parameters can be learned using supervised learning techniques.

3.1.1 Configuration Space

The configuration spaces in which harmonic functions are used to create potential field are usually discretized representations of a bounded environment. Objects, goals and the agents configurations are transformed into this space. The agent is represented as a point in this C-space. This representation makes the process of path planning simpler by avoiding intricacies in how agents are configured in the real world. This makes the implementation of the path planning algorithm to generate policies for the agent for this C-space less complex.

3.1.2 Harmonic Functions

The harmonic function used here to calculate the potential at a node computes a weighted average over its neighbors which always assure a positive and negative gradient for every node unless the agent reaches a saddle point. Such points in the grid have zero gradient and the agent searches in its neighborhood for an exit path. Let $\phi(x_i, y_j)$ be the harmonic function that represents the potential value of a node located at x_i, y_j and $w_n(x_i, y_j)$ be the weight that connects the node to its four surrounding neighbors in the X and Y direction. n is assigned a value from 1 to 4 to represent each neighbor. The equation for the modified harmonic function can then be represented as;

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

3.1.3 Weight Representation

Weights which are used in the computation of harmonic functions represent the connectivity between neighboring nodes in the discretized environment. These weights are interdependent and are modeled by functions with independent parameters to show the dependence between them. These parameters are used in later sections to capture the preferences of user motion trajectories. The potential at every node is computed by performing a weighted average over its neighbors. The four weights at each node connecting its four neighbors in the two dimensions and their sum of weights equals to one. This connection between its neighbors can be represented by the three independent parameters g0,g1,g2. The parameter g0 is used as a balancing factor between the nodes in the X and Y dimension. Parameters g1 and g2 represent the connection between neighbors in the X and Y dimension, respectively. Let w1,w2,w3 and w4 be the four weights of a node connecting each of its neighbors. The harmonic function in parametric form at x,y can then be represented as

$$\begin{aligned}\phi(x, y) = g0[&\phi(x-1, y)g1 + \phi(x+1, y)(1-g1)] \\ +(1-g0)[&g2\phi(x, y-1) + \phi(x, y+1)(1-g2)]\end{aligned} \tag{3.1}$$

where the weights are represented as,

$$
\begin{aligned}
w1 &= (g1)(g0) \\
w2 &= g0(1-g2) \\
w3 &= (1-g0)(1-g2) \\
w4 &= (1-g0)(g2)
\end{aligned}
\tag{3.2}
$$

and

$$w1 + w2 + w3 + w4 = 1$$

Solving for each parameter g0,g1 and g2,

$$
\begin{aligned}
g0 &= w1 + w2 \\
g1 &= \tfrac{w1}{w1+w2} \\
g2 &= \tfrac{w4}{w3+w4}
\end{aligned}
\tag{3.3}
$$

where,

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

The three parameters which are a function of the weights are now represented by a sigmoid function $\sigma(t)$ in order to obtain independent, unconstrained parameters. $\sigma(t) = \frac{1}{1+e^{-t}}$ where, e^{-t} is an exponential function and t is a sigmoid constant which gets updated by the algorithm. The value of this sigmoid function converges to 0 as the sigmoid parameter $t \to -\infty$ and to 1 as $t \to \infty$. The updates made to the parameters are independent of each other and do not result in a conflict in minimizing the error function. The sigmoid function maintains the parameter constraint and also takes care of normalization of the parameters.

$$
\begin{aligned}
g0 &= \sigma(t_{g0}) = \tfrac{1}{1+exp(-t_{g0})} \\
g1 &= \sigma(t_{g1}) = \tfrac{1}{1+exp(-t_{g1})} \\
g2 &= \sigma(t_{g2}) = \tfrac{1}{1+exp(-t_{g2})}
\end{aligned}
\tag{3.4}
$$

where,

$$-\infty \leq t_{g0}, t_{g1}, t_{g2} \leq \infty$$

3.1.4 Potential Field Relaxation

The successive over-relaxation technique is used to solve the homogeneous system of linear equations. This technique is based on the Gauss-Seidel iteration but converges much faster by using an exploration factor c, where $0 \leq c \leq 2$. This exploration constant is used to accelerate the approximation as shown in the recurrence relation below. In the following equation k represents the iteration number and c represents the exploration constant.

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

The exploration factor c is a constant that accelerates the convergence of the approximation of the desired potential value. To anticipate the increase in potential value at a node in future iterations, this constant is used to raise its current potential by an exploration factor. The approximation jump causes the potential value of its neighbors to increase and in turn raising it own value resulting in a lesser number of iterations required to converge. There are possibilities in which the grid does not get completely relaxed using SOR because of the residual precision, resulting in local minima or maxima. Higher values for the exploration factor could sometimes lead

to an oscillation if the desired potential value is overshot. For the exploration factor c=1, the recurrence relation for SOR reduces to a Gauss-Seidel iteration relation. The iteration is terminated if the magnitude of the update falls below a specified residual value. This residual can be arbitrarily small depending upon the floating point precision of the processor

3.1.5 Gradient Computation

The policy generated by the harmonic function path planner follows a gradient computed over the potential field. The gradient provides direction for motion from virtually every point in the environment. Since there is no local minimum, minima only exist at the goal. The negative gradient of the potential field is computed to connect the agent to the goal because goals are set to a minimum and obstacles at maximum. Let $\vec{\pi}_{x,y}$ be the gradient at a particular point (x, y) on the grid, the gradient is computed by finding the difference between its neighbors in each dimension divided by twice the square of the distance between them. Neighboring potential values. In Dimension X $\phi(x_{i-1}, y_j); \phi(x_{i+1}, y_j)$ and in Dimension Y $\phi(x_i, y_{j+1}); \phi(x_i, y_{j-1})$

Then its gradient $\vec{\pi}_{x,y}$ is computed as

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

where Δ is the internodal distance between nodes The path planner guides the agent greedily along the negative gradient of the potential field in order to reach the goal.

3.1.6 Limitations of the Path Planner

The gradient of the potential field changes only if an update is made to the environment. This occurs when new obstacles are added into the environment, a

new goal is assigned to the agent or if an obstacle changes its location dynamically. However, the gradient computed remains constant for a static environment in which no updates occur. Changes can be made to the gradient by using different boundary conditions but this could compromise safety if non standard boundary conditions are chosen and will not be able to generate a wide range of complete trajectories.

This results in a limitation of unexplored possible paths that exist from the agents current location. As the gradient generated by the planner is generic for the given local environment configuration, it can not address situations that may force the agent into taking different paths based upon preferences. Some of these preferences could be avoiding damp terrains, taking the shortest path to the goal, maintaining maximum distance from obstacles or taking smoother curves around corners. There are various other factors driving the agent along different desired trajectories which can not be captured by the original implementation of the path planner.

The standard harmonic function used for computing the potential values computes an average over its neighbors. This function gives an equal weight to each of its neighboring node potentials [3]. If an agent is surrounded by an obstacle and a goal in the X dimension then each neighbor imposes an equal force of attraction and repulsion on that node. Two different methods for weight modification were analyzed in this research which breaks the symmetry between weights since asymmetric weights are used to generate different paths. In [2] asymmetric weights have been used in the form of bias vectors to modify paths. The approach taken however still maintains symmetry between weights in different dimension resulting in a subset of paths that can be generated. These weight modifications are the same over the entire local environment, ie. every cell has the same set of weights connecting itself to its neighbors. Research work done in [1] demonstrates how weights for each cell can be altered through policy iteration to achieve the desired path. These asymmetric

weights were modified and updated by performing a gradient descent on the error between the desired and computed path. This error is only obtainable for cells along the path and thus, a very local assumption was made for the influence of weights on the error function. In this thesis research work a different local assumption for weight influence has been made and a method addressing how the error can be propagated to every cell in the local environment is added. Also, other techniques were analyzed to change the gradient computed by a harmonic function path planner. In particular artificially created sandpits that impose constraints on the relaxation of potential values for pre-selected nodes were used and the effect on path modifications was analyzed.

3.2 Sandpits

Sandpits manipulate the potential field of the environment by fixing the lower limit of a nodes potential value, thus creating the potential for artificial highlands that modify the gradient computed. They can be manually created in a given local environment in order to modify the computed gradient by generating local maximas in the potential field with no local minimas. Sandpits could be used to capture negative preferences like preventing an autonomous vehicle from navigating through damp muddy terrains or a semi-autonomous wheel chair avoiding motion on uneven rough surfaces.

For a sparse environment containing a goal positioned at the center, it is seen that the generic path computed by the path planner drives the agent in a straight line ahead to the goal. Different starting positions of the agent show the tendency of the agent to drive along the center as seen in Figure 3.1. Sandpits, when positioned in the middle as seen in Figure. 3.2 raise the potential value at the center of the grid because the released potential value does not fall below the specified threshold. The

negative gradient followed by the agent forces it to navigate along either side of the pit depending on its location from the goal. The effects of creating sandpits cannot be easily modeled in denser environments because the gradient computed over local regions is extremely shallow and setting a lower limit on the potential of specified cells may sometimes have no desired effect. Transferring sandpits to environments which are potentially similar could have undesirable effects since the sandpit cells and potential relaxation constraints cannot be directly mapped.

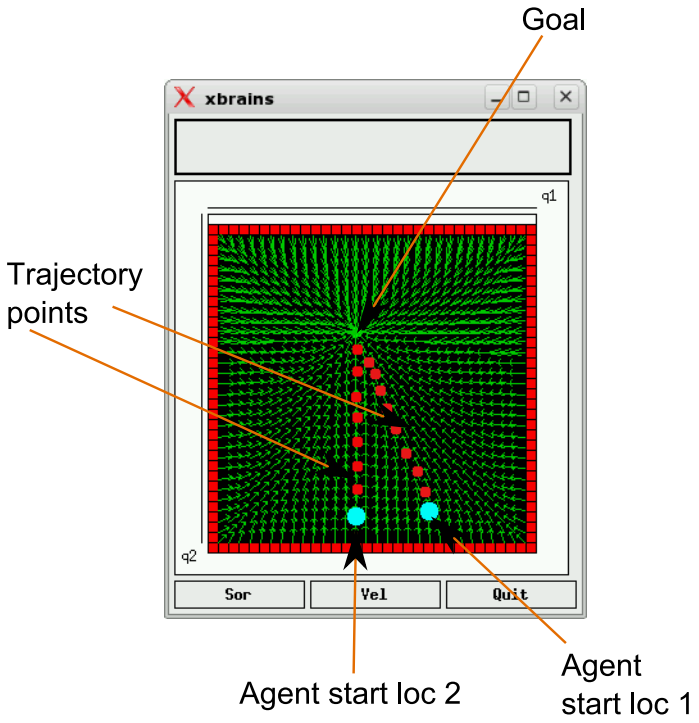

Figure 3.1. Path taken without Sandpits.

Using Dirichlet's boundary, each node or cell is assigned a potential value between 0 and 1. To create sandpits, a threshold $l(x_i, y_j)$,$0 \leq l \leq 1$ for sandpit nodes (x_i, y_j) is specified to constrain the minimum of the corresponding node's potential

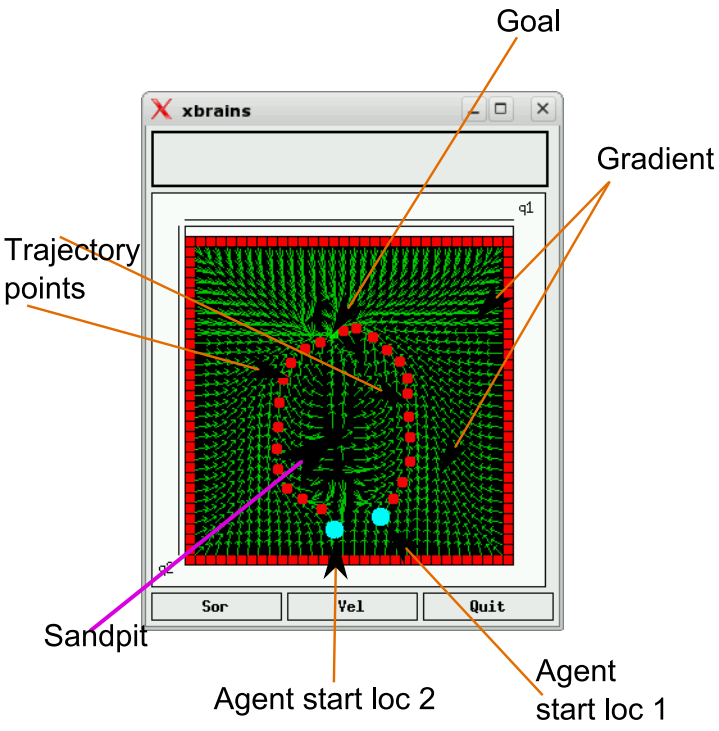

Figure 3.2. Path modification using Sandpits.

value. Let $\phi^{(k+1)}(x_i, y_j)$ be the potential to be relaxed until the $k+1$ iteration for the sandpit node located at (x_i, y_j) given the lower limit l, the potential is then computed as,

$$\begin{aligned}
\phi'^{(k+1)}(x_i, y_j) = \phi^{(k)}(x_i, y_j) \\
+\frac{c}{4}(\phi^{(k+1)}(x_{i-1}, y_j) * w_1(x_i, y_j) \\
+\phi^{(k)}(x_{i+1}, y_j) * w_2(x_i, y_j) \\
+\phi^{(k+1)}(x_i, y_{j+1}) * w_3(x_i, y_j) \\
+\phi^{(k)}(x_i, y_{j-1}) * w_4(x_i, y_j) \\
-4 * \phi^{(k)}(x_i, y_j))
\end{aligned}$$

$$\phi(x_i, y_j) = max(l(x_i, y_j), \phi'^{(k+1)}(x_i, y_j))$$

3.3 Modifying Weights

To modify the control policy such that it matches the gradient direction of the user's trajectory, the symmetry between weights needs to be relaxed. A node's potential is computed by taking an average over its neighboring values. The harmonic function can be modified to calculate a nodes potential value by taking a weighted average over its neighbors. A higher weight value between two nodes indicates a stronger connection between them and vice versa. The stronger the connection between nodes, the larger the force exerted. For example, the agent could prefer to move closer to the right of an empty hallway when exiting, but move along the left wall when entering. If all nodes exert an equal force then the agent is forced to drive along the center of the hallway as seen in Figure 3.3. Modifying the weight connections between the left and right neighboring nodes produces a change in trajectory. The change is produced by allowing the left wall of the hallway to exert a greater force of repulsion causing the agent to drive along the right side as seen in the Figure 3.4

This modified behavior counteracts the natural tendency of the agent to move along the center of the hallway caused by both walls of the hallway exerting equal force. The repulsive force exerted by obstacles still exists but the effect varies. Hence, the policy generated by the modified planner still maintains all the characteristics of a good path planner. Modeling the effects of weight modifications on the gradient is comparatively easier for a sparse than a dense environment. Figure 3.5 shows that the control policy generated for an agent to navigate down a crowded hallway by modifying weight connections can not be easily understood. In the next section we explain in detail how the weights are adjusted to modify the path generated by a harmonic function path planner.

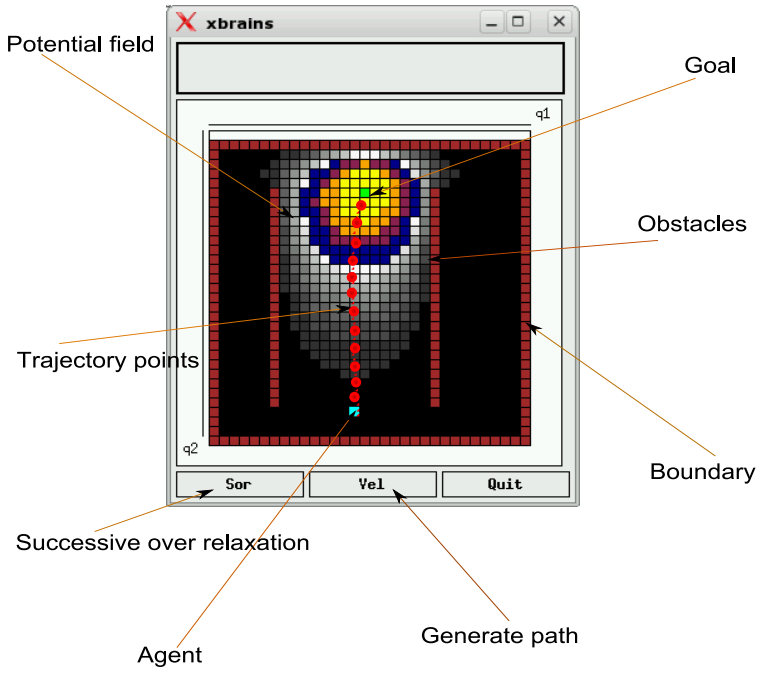

Figure 3.3. Generic trajectory.

3.4 Learning Weights

Weight modifications can be made to relatively small regions of the grid to better predict the behavior of the agents motion in that region. The effects of weight modification in small local regions may influence other regions by a magnitude which is hard to formalize. Weights should then be modified such that the global effect on its local environment is taken into account. These modified weights are adjusted iteratively until the control policy matches the reference policy. This is the learning phase of the algorithm in which the planner tries to minimize an error function.

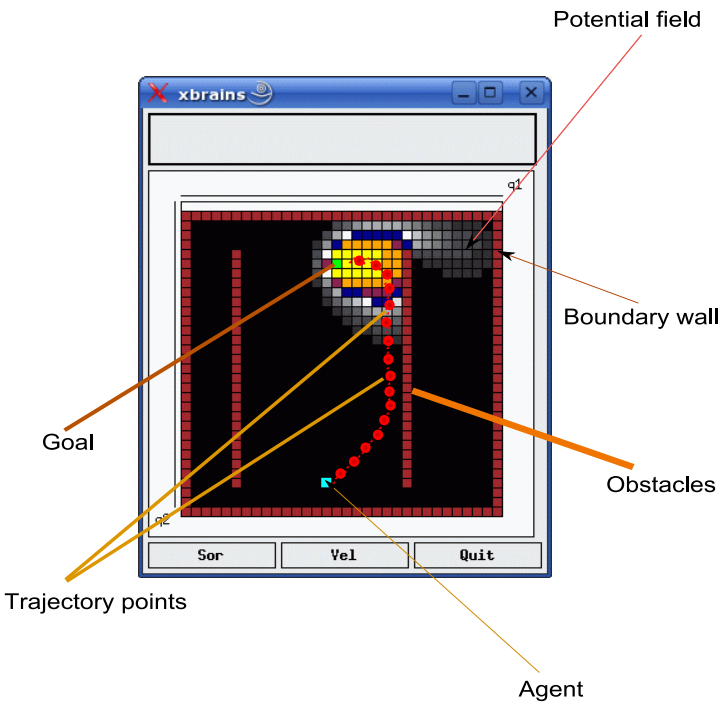

Figure 3.4. Path modification using weights.

The error function e is chosen such that the angle between the gradient direction of the current policy and the reference policy is minimized. The error function takes the sum over all errors of nodes that lie along the user path. The reference policy $\vec{\nabla}P$ is extracted from user trajectories and $\vec{\pi}$ is the gradient computed over the current potential field. The learning phase terminates when the error function e where $0 \leq e \leq 1$, is reduced below a specified threshold. The third power is considered in order to magnify large errors while performing a gradient descent on the error.

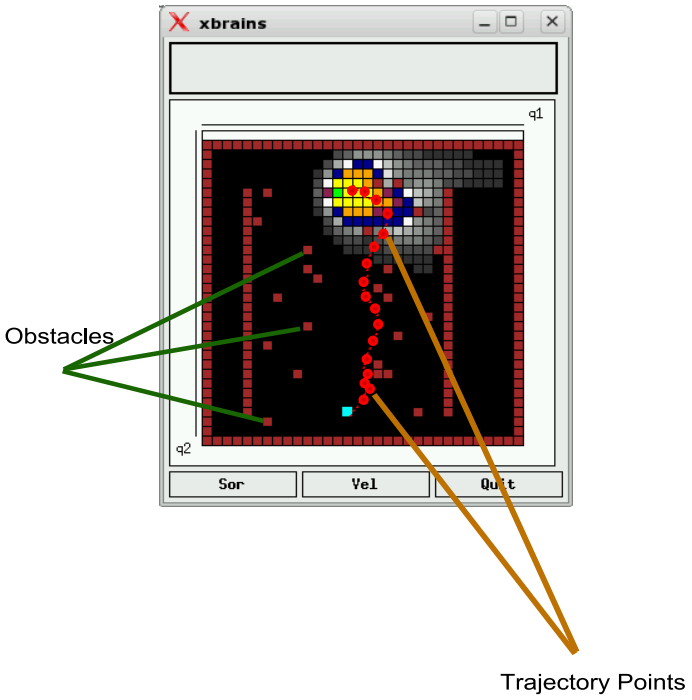

Figure 3.5. Navigation in dense environment.

$$e = \sum_{node \in path} (1 - cos(\vec{\pi}, \vec{\nabla} P))^3 \tag{3.5}$$

The measure of influence a node weight has on the error function depends on its distance from a path node. To find the weights that influence the potential at a given node, a local approximation template is designed. This template is placed over each node to find the nodes whose weights have direct influence on the given node's potential. This template is a local approximation of the global effect of weights that influence the error at a node. The template is designed such that it covers all

node weights which are less than or equal to three steps away from the selected node. This is a local approximation model since all node weights of the local environment have an influence on the node's potential but their effect decrease exponentially with distance, thus node weights at a distance of three are fixed here.

The amount of change in error produced by these weights is then calculated. The measure of influence is determined by taking the derivative of error with respect to each weight provided by the template. Each weight directly or indirectly influences the potential at more than one node and thus the derivative of potential with respect to weights can be expressed as

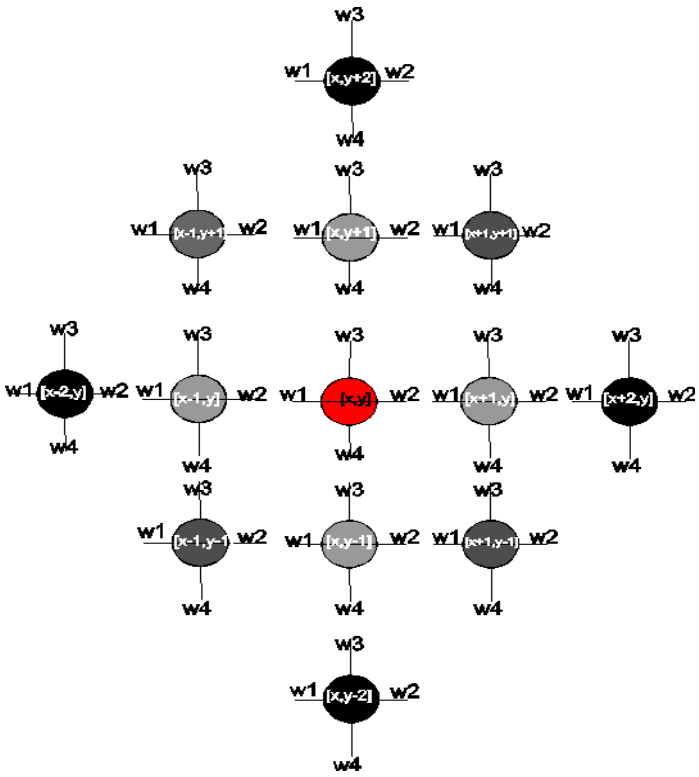

Figure 3.6. Weight influence template.

$$\frac{\partial \phi_q}{\partial W} = \sum_{i \in N(q)} \frac{\partial \phi_q}{\partial w_i} \tag{3.6}$$

Where,

$$q = (x, y) \quad N(q) = \{w_i(x_a, y_b) | (x_a, y_b) \in N(x, y)\}$$

$N(q)$ is the neighborhood defined by the template placed on the node located at q and w_i are all node weights covered in this template. The template seen in Figure. 3.6, shows all weights that influence the potential of the node located at (x, y).

The information provided by the reference policy contains only direction vectors for nodes along the path traversed by the user. Hence only weights surrounding these nodes would get modified according to the template. To address this, the influence of all weights in the grid on the error function is modeled to get an accurate approximation to the reference control policy. The error at the nodes along the user's trajectory therefore needs to be propagated outwards to other nodes, where no directional error is available.

3.4.1 Policy Optimization

The algorithm below adjusts the set of weights W, to generate a policy $\vec{\pi}$, that approximates a user's trajectory. These weight modifications are performed iteratively until the error lies below a specified threshold e^0. Each weight is represented by three independent parameters g0,g1,g2. Each of these parameters is constrained to values between 0 and 1. These weights are represented by independent parameters to model the explicit inter-dependence between weights. Each parameter is then modeled by a sigmoid function. Sigmoid functions are used because of its non-linearity in assuring that the parameter values for g0 ,g1 and g2 remain within the specified

range. The effect of these sigmoid parameters t_{g0}, t_{g1}, t_{g2} on the error function is then calculated to indirectly update the weights.

The algorithm below modifies weights to minimize the error function by computing the derivate or the error with respect to the sigmoid parameters. Weights are updated and the new gradient is computed. This process continues until the error falls below a specified threshold.

The algorithm described in the following steps tries to minimize the error function e.

$$e = \sum_{q \in N(i)} (1 - cos(\vec{\pi}, \vec{\nabla} P))^3$$

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* e^0*)* **do**

Compute $\frac{\partial e}{\partial \phi}$ for all nodes using error propagation

Compute the gradient $\vec{\nabla} e = \frac{\partial e}{\partial \vec{t}} = \sum \frac{\partial e}{\partial \phi} . \frac{\partial \phi}{\partial \vec{t}}$

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 adjustments are computed by taking the derivative of the error with respect to each of these sigmoid parameters. To find this update, the derivative of error with respect to each of these sigmoid parameters is first calculated.

$$\vec{\nabla} e = \frac{\partial e}{\partial \vec{t}} = \sum \frac{\partial e}{\partial \vec{\pi}} . \frac{\partial \vec{\pi}}{\partial \phi} . \frac{\partial \phi}{\partial weight} . \frac{\partial weight}{\partial parameter} . \frac{\partial parameter}{\partial \vec{t}} \tag{3.7}$$

where,

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

3.4.2 Potential Error Propagation

An algorithm similar to the backpropagation algorithm is used to propagate the directional error outward from the nodes whose directional error is known. Since no directional error is available for other nodes in the region, the potential error is propagated in the outward direction to these nodes. These nodes act similar to the hidden nodes of a feed forward neural network. The error is propagated backward one step at a time to avoid recursive dependence. Manhattan distance is used to label nodes to represent the structure similar to the feed forward network. In the first iteration every neighbor of a path node gets labeled 1. In the second iteration all nodes that are neighbors to the nodes labeled 1 are incremented by one and so on until all neighbors get labeled. If path nodes are neighbors to other path nodes they get labeled 1, else they would be labeled 2. Nodes that are marked as goals or obstacles are labeled -1. 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 zero. The potential error is computed sequentially in accordance with its distance from the path nodes. For example the error is first propagated to all nodes labeled 2 before propagating them to nodes labeled 3.

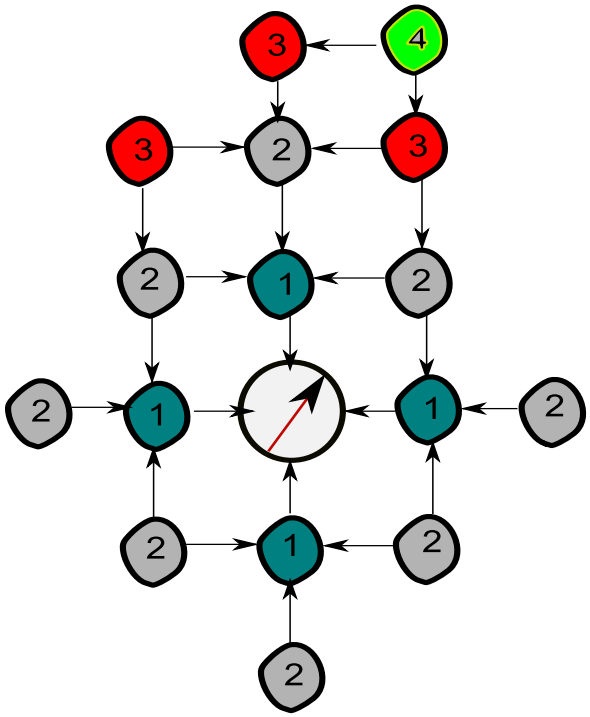

Figure 3.7. Network for error propagation.

The potential error at a node is calculated by finding the derivate of the error with respec to to its potential. The derivatives for each node are initialized to zero. 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 have a directional error. The change in potential at nodes labeled 1 have a direct influence in minimizing or maximiming the directional error of its neighbors.

The derivative of the error function with respect to its potential ϕ_i where $i = 1$ is calculated as follows [1],

$$\frac{\partial e}{\partial \phi_i} = \sum_{j \in M(i)} \frac{\partial (1 - cos(\vec{\pi}, \vec{\nabla} P))^3}{\partial \phi_j}$$
$$= \sum_{j \in M(i)} -3(1 - cos(\vec{\pi}, \vec{\nabla} P))^2 . \frac{\partial}{\partial \phi_j} [\frac{\vec{\pi}.\vec{\nabla} P}{|\vec{\pi}||\vec{\nabla} P|}] \tag{3.8}$$

Hence,

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

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

$$\phi_j|_q = \frac{\phi_{q-} - \phi_{q+}}{2\Delta^2}$$

$$\frac{\partial \vec{\pi_j}}{\partial \phi_i} = \frac{1}{2\Delta^2} [\frac{\partial \phi_{q-}}{\partial \phi_i} - \frac{\partial \phi_{q+}}{\partial \phi_i}]$$

Where, Δ is the internodal distance.

M(i)is the neighbourhood under the feed-forward structure of the current node which is labeled 1, they are implicitly the four neighbors in the X and Y dimension. ϕ_i is the potential at i, $\vec{\nabla} P$ is the optimal control policy we are trying to match, $\vec{\pi}$ is the control policy we modify to achieve a match.

As seen in Algorithm 1, the error is propagated outward to all nodes which are at a distance greater than 1 from a path node. In order to compute the derivative of the error function with respect its potential, the gradient direction of at least one of its neighbors is required. Since the gradient direction is not available for nodes at a distance greater that 1, the derivative of the error with respect to nodes labeled 1 are backpropagated outward. The derivative error with respect to the potential ϕ_i, where node i has a label $q \geq 2$ can then be computed using the equation,

$$\frac{\partial e}{\partial \phi_i} = \sum_{j \in M(i)} \eta_j . \frac{\partial \phi_j}{\partial \phi_i}$$

Where,

$$\eta_j = \frac{\partial e}{\partial \phi_j}$$

3.4.3 Computing Parameter Adjustments

From 3.7 the parameter space gradient $\frac{\partial \phi}{\partial t}$ can be reduced to Equation. 3.9.

$$\frac{\partial \phi}{\partial t} = \frac{\partial \phi}{\partial \vec{w}} \frac{\partial \vec{w}}{\partial \vec{p}} \frac{\partial \vec{p}}{\partial \vec{t}} \tag{3.9}$$

where,

$$\vec{w} = (w1, w2, w3, w4)^T$$

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

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

$\phi(x, y)$ is a function of its weights hence its partial derivative with respect to its own weights can be easily computed. The derivative with respect to weights which indirectly influence the potential at (x, y) according to the local template are also considered during the computation. Since $\phi(x, y)$ is computed as a weighted average over its first neighbors, its derivative with respect to the weights within the local template can be computed.

Given,

$$\phi(x, y) = \phi(x-1, y)w1 + \phi(x+1, y)w2 + \phi(x, y-1)w3 + \phi(x, y+1)w4$$

the derivative of the direct weights of the node at the center of the template can be computed as

$$\frac{\partial \phi}{\partial w1} = \phi(x-1, y) \quad \frac{\partial \phi}{\partial w2} = \phi(x+1, y) \quad \frac{\partial \phi}{\partial w3} = \phi(x, y+1) \quad \frac{\partial \phi}{\partial w4} = \phi(x, y-1)$$

The derivative with respect to weights of nodes within the local template at distance greater than 0 can then be computed indirectly. Let w_z be the weight of

node located at z and where z is at a distance d from (x, y) within the local template. For d at a distance the equation can be expressed as

$$\frac{\partial \phi(x,y)}{\partial w_z} = \frac{\partial \phi(x,y)}{\partial \phi(q)} \frac{\partial \phi(q)}{\partial w_z}$$

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 Equations 3.1 to see how the potential can be expressed in terms of the parameters. On solving for Equation 3.1 weights can be expressed as a function of the parameters as seen in Equation 3.2. The partial derivate is then computed as follows:

$$\frac{\partial \vec{w1}}{\partial g0} = g1 \quad \frac{\partial \vec{w2}}{\partial g0} = 1 - g1 \quad \frac{\partial \vec{w3}}{\partial g0} = g2 - 1 \quad \frac{\partial \vec{w4}}{\partial g0} = -g2$$

$$\frac{\partial \vec{w1}}{\partial g1} = g0 \quad \frac{\partial \vec{w2}}{\partial g1} = -g0 \quad \frac{\partial \vec{w3}}{\partial g1} = 0 \quad \frac{\partial \vec{w4}}{\partial g1} = 0$$

$$\frac{\partial \vec{w1}}{\partial g2} = 0 \quad \frac{\partial \vec{w2}}{\partial g2} = 0 \quad \frac{\partial \vec{w3}}{\partial g2} = g0 - 1 \quad \frac{\partial \vec{w4}}{\partial g2} = g0 + 1$$

As seen previously parameters have been represented as sigmoid functions to constrain their value between 0 and 1 as seen in Equation 3.4. Hence the last term from Equation 3.9 can be computed as.

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

Hence,

$$\frac{\partial g0}{\partial t_{g0}} = g0 - (g0)^2$$

$$\frac{\partial g1}{\partial t_{g1}} = g1 - (g1)^2$$

$$\frac{\partial g2}{\partial t_{g2}} = g2 - (g2)^2$$

3.4.4 Weight Update

After each iteration within the policy optimization process weights are updated to reflect changes in the gradient over the new potential field. Since these weights are represented in a parametric form they are indirectly updated by first updating their representing parameters. These parameters modeled as sigmoid functions are updated as follows

$$g0 = \sigma(t_{g0}) = \frac{1}{1 + exp(-t_{g0})}$$
$$g1 = \sigma(t_{g1}) = \frac{1}{1 + exp(-t_{g1})}$$
$$g2 = \sigma(t_{g2}) = \frac{1}{1 + exp(-t_{g2})}$$

Where the sigmoid parameters were initially updated by performing a gradient descent on the error function.

$$t_{g0} \leftarrow \Delta t_{g0} + t_{g0}$$
$$t_{g1} \leftarrow \Delta t_{g1} + t_{g1}$$
$$t_{g2} \leftarrow \Delta t_{g2} + t_{g2}$$

The weights for a node at q which are expressed as a function of the parameters are now updated with the newly computed parameters as follows,

$$w1_q = (g1)(g0)$$

$$w2_q = g0(1 - g2)$$

$$w3_q = (1 - g0)(1 - g2)$$

$$w4_q = (1 - g0)(g2)$$

CHAPTER 4

EVALUATION

4.1 Setup

A simulator that was originally built for planning the motion of an agent in C-space is extended to analyze the generation of customized policies from a set of preferences. The simulator was built in C under the 32-bit Linux platform and tested on a 32-bit Intel Pentium 4 processor with a CPU speed of 2.4GHz. The grid map of the environment considered for the experiments is a discretized C-space representation of a local environment. The C-space space for this environment is enclosed within an artificially created boundary in order to relax the potential. This grid map is visually interactive and can be manually designed using a combination of keyboard and mouse inputs. The agent, obstacles and goals are all transformed into this simplified bitmap representation. In this C-space the agent is represented as a point and all obstacles and goals are marked by grid cells. This grid map of the environment is composed of evenly-spaced cells. The size of the grid is 32×32. A harmonic function path planner is used for generating the control policy which uses Dirichlets boundary conditions to generate smooth, safe paths that are complete and correct. According to Dirichlet's condition the obstacles are set at a fixed maximum potential. 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. The two functionalities provided by the planner are (SOR) to relax the potential field and (VEL) to compute the trajectory of the agent from the start location to the goal.

4.2 Implementation

4.2.1 Overview

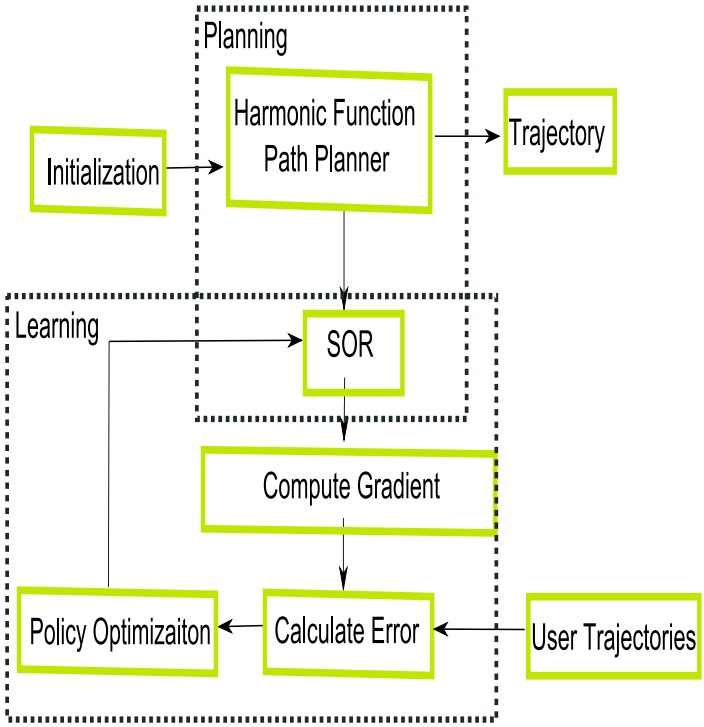

Figure 4.1. Desired path genereation overview.

The Figure. 4.1 shows an overview of the learning and evaluation process to illustrate the working of the implementation. The process of learning weights is an iterative process in which weights are modified and grid potential re-relaxed to reflect changes in the computed gradient. At every iteration the error is computed to to see if the user trajectory is approximated, if the error is above a specified threshold the iterative process continues.

4.2.2 Initialization

In this phase, the environment is setup where the obstacles, goals and the agent's position are specified and input to the planner. Each cell which represents an obstacle on the grid is assigned a potential of 1 and a potential of 0 is assigned for a goal. The connection between cells is represented by weights. This set of weights is provided to the planner from a command line. As described earlier these weights are modeled using sigmoid parameters. Motion preferences can be represented through motion trajectory points as they provide the necessary directional vectors. These directional vectors were considered to be a good measure for error computation, with respect to the gradient directions of the path planner.

A user path in a given environment is generated by selecting a set of cells on the grid leading to the goal. From these selected cells the directional vectors leading to the goal are computed. The directional is then computed as a vector from the center point of the current cell to the next cell on the path which was marked. This is continued until every cell along the path has a directional vector.

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 HFPP (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. Cell weights across the entire grid are modified. The weight modification is done by performing a gradient descent on the error function $(1 - cos(\vec{\pi}, \vec{\nabla} P))^3$. The learning rate used in updating these weights is made to be adaptive so that it updates more slowly as the error decreases in order not to overshoot.

4.2.3 Planner

The path planner is initialized with a set of weights in order to computed a path where the default weights are ($w1 = w2 = w3 = w4 = 0.25$) which are only capable of generating generic paths.

Figures. 4.2 and 4.3 show how the gradient can be modified using asymmetric weights. These were manually adjusted to drive the agent along the right wall of the hallway. Weights of $w1 = 0.35$ $w2 = 0.15$ $w3 = 0.25$ $w4 = 0.25$) were set throughout the entire grid for every grid cell, resulting in a greater force of repulsion from the left walls of the hallway.

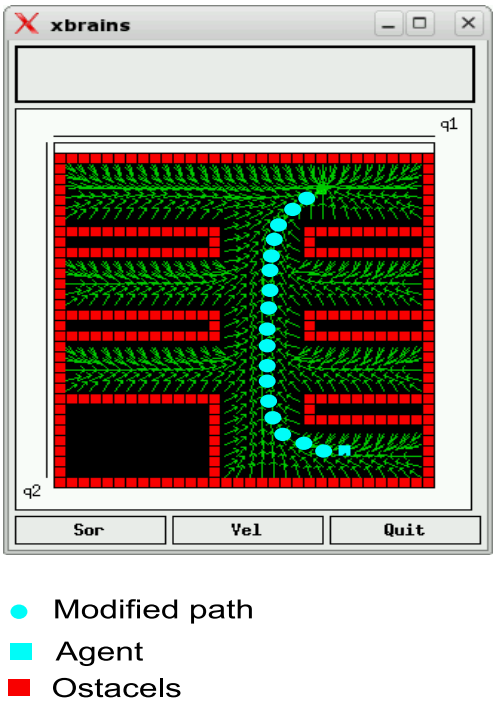

Figure 4.2. Orginal gradient.

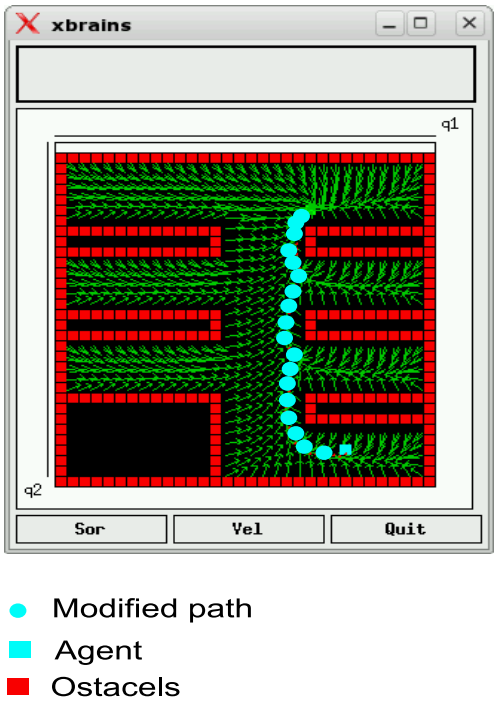

Figure 4.3. Modified gradient.

The experiments done show how these weights are automatically learned by the planning algorithm until the gradient generated matches the directional vectors of a user specified path. A symmetric weight distribution is used for the initial weights in order to learn. These default weights are then fed to the SOR module to generate a potential field. The value for the exploration factor c is set at 1.8. The cell's potentials get relaxed until all potential value updates fall below a specified residual value. The residual used is 10^{-14}. This residual depends on the floating point precision of the CPU.

Once the grid is relaxed using the updated set of weights the negative gradient is computed and the path error measured. This process continues until the error falls below a specified threshold.

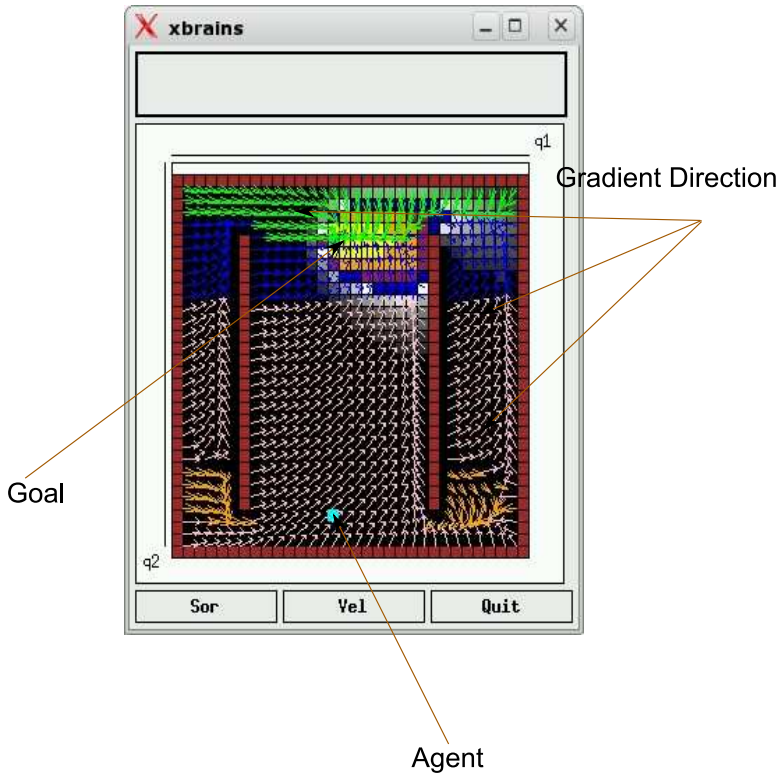

Figure 4.4. Gradient vectors over the potential field.

The negative gradient is then computed over this potential field. The potential point of every cell is located on the top left corner of the cell.

4.2.4 Policy optimization

The gradient generated by the planner is used to calculate the error function. Cosines of the angle between the reference and current directional vectors are used. Cosines are used so that difference wraps at 180 degrees. The function takes cubes of the cosines so that cells having maximum error get amplified. The negative of the sum is taken because the max error lies at -1 and the min at 1 ($cos(0)$)

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

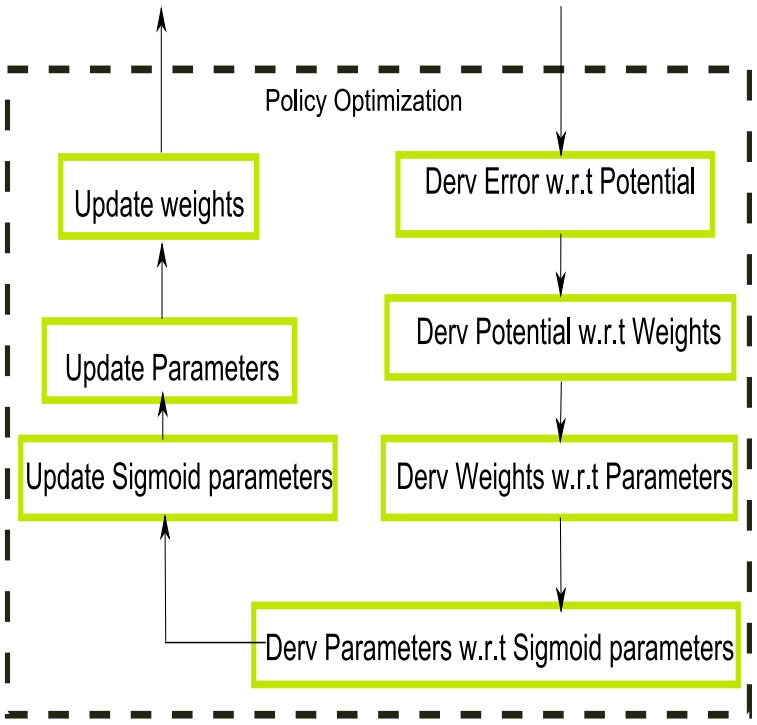

Figure 4.5. Policy optimization.

Since the directional error is only available for cells along the path $\frac{\partial e}{\partial \phi}$ is propagated outward like in a feed forward network to other cells where no error is available. The cells at every layer are marked by the Manhattan distance from the path cells containing errors. These cells (in order of increase) are then stored in a linked list for one step sequential error propagation like in a feed forward network.

The gradient descent performed on this error function minimizes this error until it falls below a user specified threshold. Computing $\frac{\partial e}{\partial t}$ gives the update required for each sigmoid parameter. Since weights are modeled as sigmoid parameters the weights are then updated.

These new weights are then fed back into SOR for a new gradient to be computed. The entire is shown by flow diagrams in Figures. 4.1 and 4.5.

4.3 Experiments

Two modes were initially considered to capture user trajectories. The first technique selects a user path from the negative gradient over a potential field created using random asymmetric weights. This method ensures that trajectories can be learned exactly and were used to evaluate the convergence properties of the learning approach. The next technique which is the prefered method for user experimentation uses hand drawn user trajectories to test the policy optimization for complex paths. These user paths approximate real paths taken by users in the real world and thus give us an accurate reference for policy optimization. Experiments performed show the minimization of the error function for different environment setups.

4.3.1 Convergence Experiments

A user path can be generated by using modified weights using the HFPP. This technique ensures that the path generated can always be approximated. In this experiment a user trajectory is generated by modifying the path planner with a random set of asymmetric weights. The weight connections for each cell in the local region was set to $w1 = 0.40,\ w2 = 0.10,\ w3 = 0.35,\ w4 = 0.15$. Figure 4.6 shows the modified path in the given environment.

It took SOR 36 itererations to relax the potential field over the entire grid, given an exploration factor of 1.8. Starting with the initial symmetric default weights, the weights were then learned and modified to approximate the user path using a learning rate of 2. After performing policy optimization for 20000 iterations the trained weights are used in creating the potential and the negative gradient is computed. Table 4.1 shows the final angle errors along the path cells in terms of their cosine

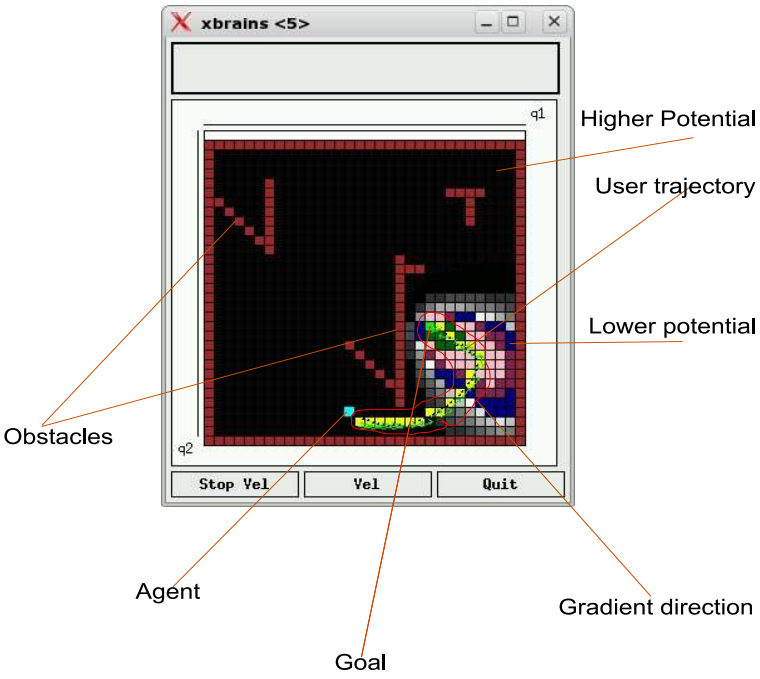

Figure 4.6. Experiment with HFFP generated path.

In this representation the minimum error is the $cos(0) = 1$ and the maximum error is the $cos(180) = -1$ and the values in the table show that the algorithm was successful at closely approximating the target gradients.

The learning curve for the error function e is shown in Figure.4.7

where

$$e = (1 - cos(\theta))^3$$

and θ is the angle between the two directional vectors and $0 \leq \theta \leq \pi$

Table 4.1. Experiment1:Difference between Trajectories

Cell Location	Error
[23][12]	0.906925
[24][11]	0.996940
[25][10]	0.993560
[26][10]	0.933675
[27][9]	0.989894
[27][8]	0.997643
[26][7]	0.999984
[26][6]	0.997561
[25][5]	0.997410
[24][4]	0.997668
[22][3]	0.998026
[23][3]	0.996772
[15][2]	0.988443
[16][2]	0.987381
[17][2]	0.990524
[18][2]	0.996381
[19][2]	0.996224
[20][2]	0.968512
[21][2]	0.996166

Initially, the error plot for the previously presented environment setup in Figure 4.3 is shown in Figure 4.8. Again it can be observed that the directional error slowly converges to a minimum.

To obtain a more reliable estimate of convergence rates for this type of experiment, the average of the error function plot for 20 different experiments is shown in Figure. 4.9. The experiments were performed with different environment setups.

4.3.2 User Experiments

To investigate the applicability of the learning approach to arbitrary target trjectories, a set of experiments were performed where user trajectories were hand

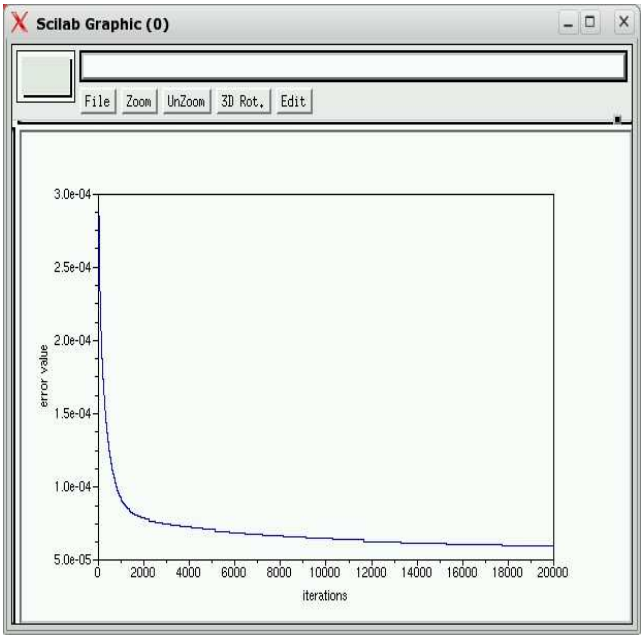

Figure 4.7. Learning curve for Experiment 1.

created. These experiments generally required very significant changes in the gradient direction.

Experiment 1: In experiment 1 it took SOR 110 msec, 1065 iterations with an average of 0.103286 msec/iter, to relax the potential to a residual error of 9.81437e-15 with an exploration factor of 1.5. Figures 4.10 and 4.11 show that the gradient field and path before and after the weight optimization, respectively. The grid direction error between the user trajectory and the gradient direction computed by the modified path planner. The error shown in Table.4.2 is in terms of the cosine of the angle between the user direction and the gradient direction, where the maximum error is at -1 and minimum at 1. The time taken for policy optimization was 86112.7sec and the average error after 200,000 iterations has been reduced to 0.012627 which is

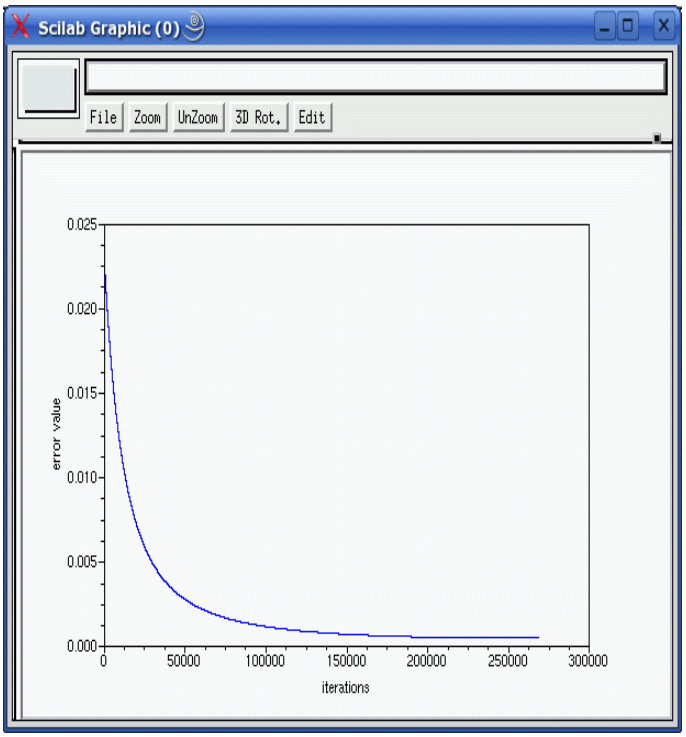

Figure 4.8. Learning curve for Experiment 2.

approximately an average error of 5deg between user direction and gradient vectors. From Table.4.2 it can be seen that all except for the begining and end pieces of the trajectory curve converging to a minimum for the cosine of the angle between the 2 vectors.

Experiment 2: In experiment 2 we see how a preference over the goal is set. There are two goals specified in this environment, the default negative gradient produced by the harmonic function path planner using symmetric weights causes the agent to move towards the goal to the top left as seen in Figure 4.12. A user trajectory is then selected to navigate the agent towards the goal at the right hand side as seen in Figure 4.13.

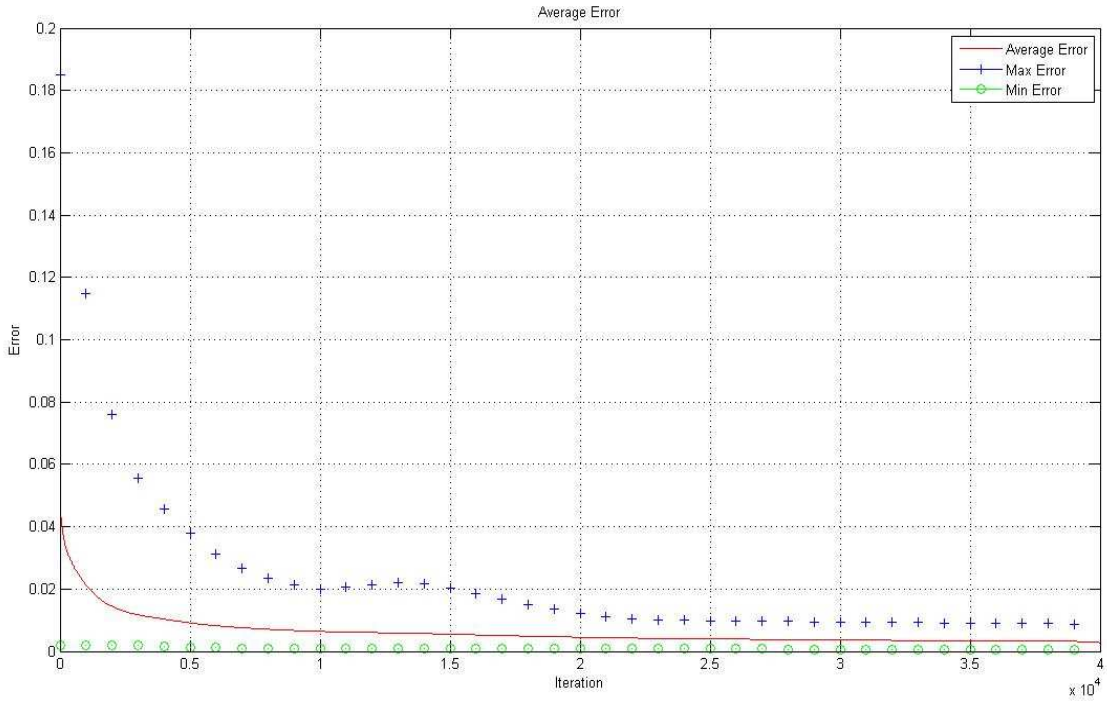

Figure 4.9. Average error plot.

The average error plot for this environment setup is seen if Figure 4.14. The minimum error after 707×50 iterations is observed as 0.000999 which is approximately 1.40deg. The direction error is shown in Figure 4.3.

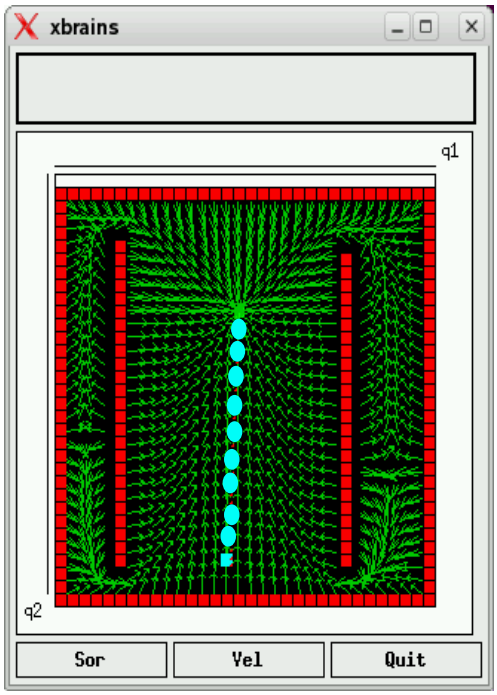

Figure 4.10. Original Gradient Experiment1.

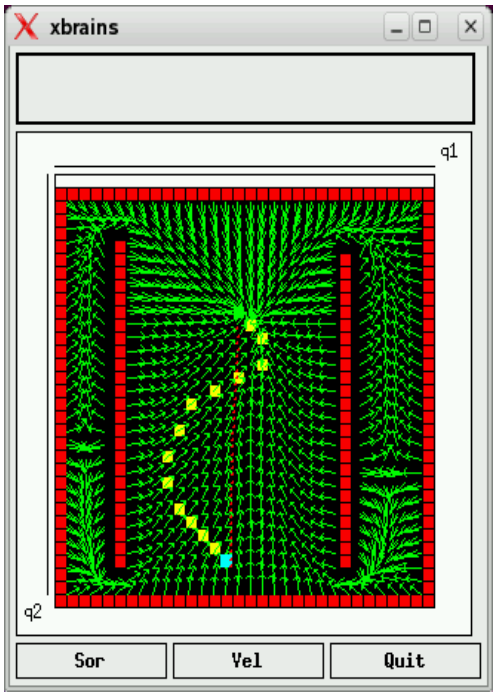

Figure 4.11. Modified Gradient Experiment1.

Table 4.2. Experiment1:Modified Grid Direction Difference

Grid Location	Angle Error
[16][21]=0.976189	
[17][20]	0.487818
[17][18	0.723625
[15][17]	0.993725
[13][16]	0.995286
[11][15]	0.973974
[10][13]	0.948317
[9][11]	0.959623
[9][9]	0.991287
[10][7]	0.990517
[11][6]	0.949683
[12][5]	0.869769
[13][4]	70.817254

Table 4.3. Experiment2:Modified Grid Direction Difference

Grid Location	Angle Error
[22][11]	0.999999
[23][11]	0.995047
[24][11]	0.949950
[21][10]	0.981407
[20][9]	0.952910
[19][8]	0.933831
[18][7]	0.920779
[17][6]	0.910326
[16][5]	0.903901
[15][4]	0.805519

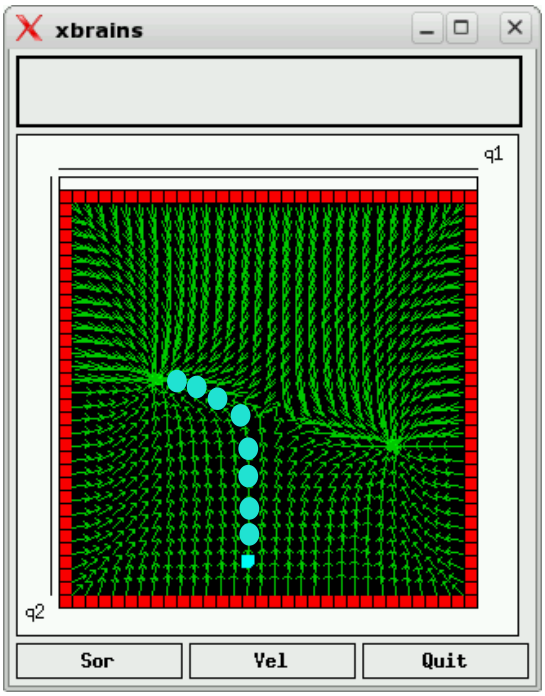

Figure 4.12. Orginal Gradient Experiment2.

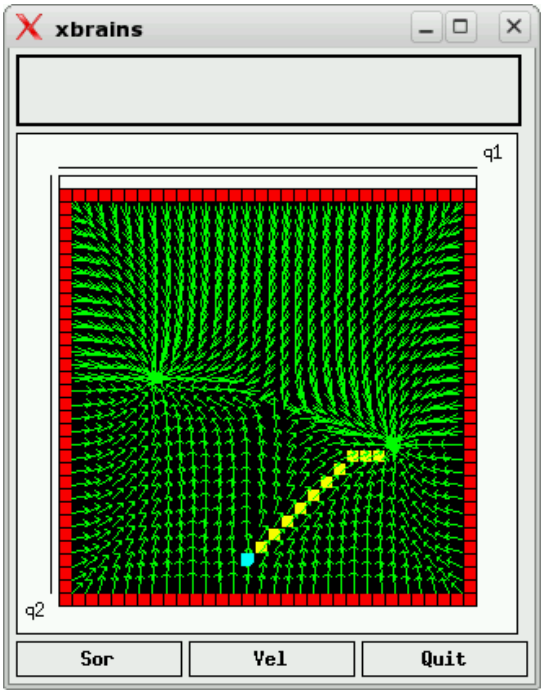

Figure 4.13. Modified Gradient Experiment2.

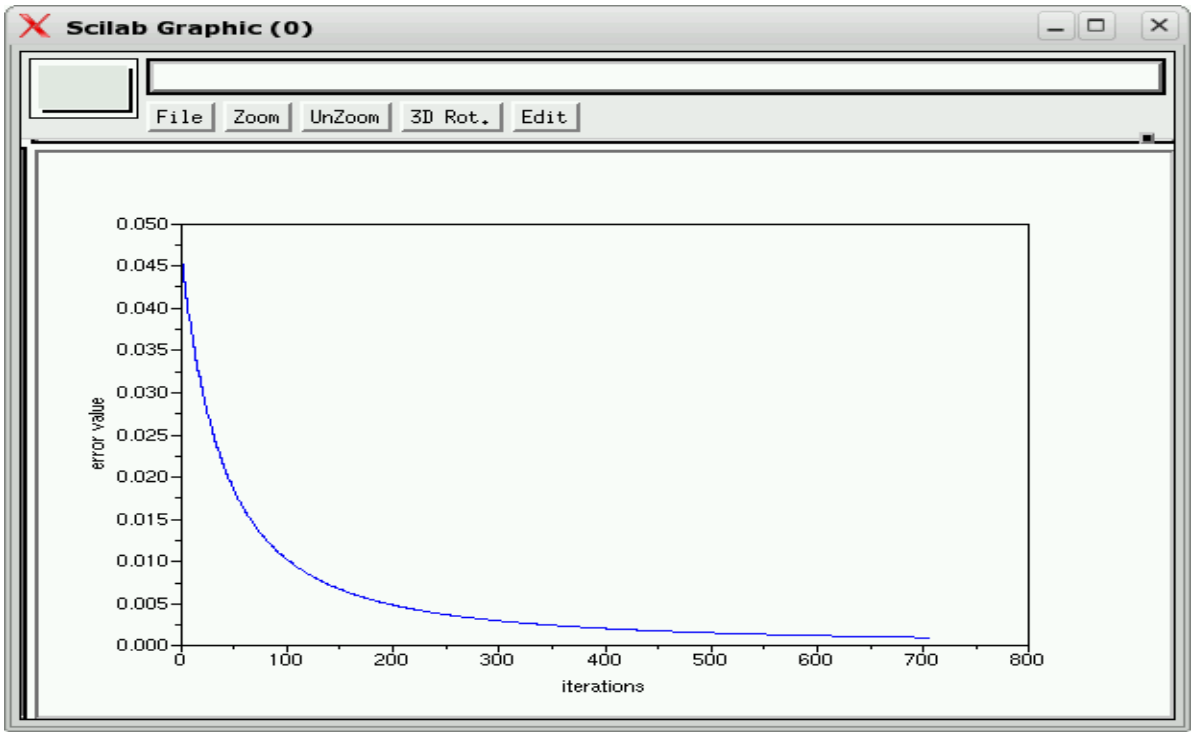

Figure 4.14. Learning curve for Experiment2.

CHAPTER 5

CONCLUSION

This chapter provides a summary of the techniques that were used, the results that were achieved and its application in different domains. The final discussion is for possible future work for improvements in the framework and extensions to integrate with other systems.

5.1 Final Thoughts

Harmonic functions exhibit useful path properties by generating smooth, complete and correct paths that have no local maxima or minima. Path planners designed using harmonic functions are useful for robust path planning in real time environments as they can be easily computed for a given local region. Therefore, harmonic functions were an ideal choice to analyze path planning customizations. Many researchers have previously used harmonic functions for path planning but only a limited amount of work was done towards path modification and customization as seen in [2] [1]. Even these approaches had restrictions on the various different paths that could be generated. All of this work was limited to optimizing global properties and not customizing to individual paths.

The motivation of this research came from the ability of the harmonic function path planner (HFPP) to generate customized paths based on user trajectory preferences. The planner was designed to go beyond producing just simple generic paths to more diverse, user specific paths. Successful experimental results of path modification (by the HFPP) have shown that various different paths can be generated to reach

the same target in a given environment. The goal was to represent user trajectory preferences as harmonic function parameters to generate these customized paths and to later on be able to transfer these preferences to potentially similar environments. The paths generated in these new environments are always assured of being complete and correct. The HFPP not only approximates the user trajectory but also generates a control policy for the entire local environment and preserves the transferred preferences. This is achieved by the negative gradient computation over the entire potential field providing directional vectors leading to the goal from virtually every single point in the region.

Following from the above motivation the harmonic function path planner was modified by changing their parametric representation. The change in parameters helped in understanding how simple generic paths could be modified. These results helped analyze the influence these parameter changes had on the path. Analyses were done to find how these parameters could be modified to generate customized paths. HFPP parameter modification required user trajectory as a reference of how the desired path should look. Previous trajectory samples that a user would realistically take in a given environment configuration were considered as examples of preferences. These trajectories were different from the generic paths generated by the unmodified HFPP and provide directional vectors that are used to measure the error between it and the gradient direction computed by the HFPP. A local assumption was then made to model the influence of parameters on the directional error. Since these directional errors were only available for cells along the reference user path, error propagation techniques were used to better model the influence of every parameter in the local region with respect to the error.

The HFPP policy was then iteratively modified until the desired user trajectory was approximated. The results show that by choosing the right error function

and performing a gradient descent on the error the desired trajectory can be closely approximated through HFPP parameter modification.

Various complex user paths were considered as performance indexes for our policy optimization algorithm and results observed have shown closely matching paths produced by HFPP.

The modified HFPP can be used to learn and transfer user trajectory preferences to new unexplored environments with similar configurations and pattern. The HFPP can be integrated to control the motion planning of semi-autonomous wheelchairs, remote controlled mobile robots, robotic arms, intelligent game characters, etc

5.2 Future Work

Some paths could be extremely complex for the HFPP to learn. These limitations could be addressed by studying the influence of integrating sandpits with HFPP parameter modifications.

Not every motion preference can be captured from user trajectories alone; it may require learning user personalities, behaviors and conditions. Transferring motion preferences is the ultimate desire for HFPP modification. This research work does not include the finding of matching local patterns in environments for the transfer of motion preferences. Various machine learning algorithms could be used to categorize these environments. Semi-autonomous wheelchairs would be an ideal test bed for analyzing the performance of the HFPP.

REFERENCES

[1] J. A. Coelho, R. Sitaraman, and R. A. Grupen, "Parallel optimization of motion controllers via policy iteration."

[2] M. A. I. Fabio Dapper, Edson Pretes and L. P. Nedel, "Simulating pedestrian behavior with potential fields." Springer Berlin / Heidelberg, 2006, pp. 324–335.

[3] C. I. Connolly and R. A. Grupen, "On the applications of harmonic functions to robotics," *Journal of Robotic Systems*, vol. 10, pp. 931–946, 1993.

[4] J. Borenstein, Y. Koren, and S. Member, "The vector field histogram - fast obstacle avoidance for mobile robots," *IEEE Journal of Robotics and Automation*, vol. 7, pp. 278–288, 1991.

[5] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *Int. J. Rob. Res.*, vol. 5, no. 1, pp. 90–98, 1986.

[6] S. M. L. Valle, *Planning Algorithms*. Cambridge University Press, 2006.

[7] G. Song and N. M. Amato, "Using motion planning to study protein folding pathways," in *Journal of Computational Biology*, 2001, pp. 287–296.

[8] S. Thrun and A. Buecken, "Integrating grid-based and topological maps for mobile robot navigation," in *Proceedings of the AAAI Thirteenth National Conference on Artificial Intelligence*, 1996.

[9] D. Rodríguez-Losada, F. Matía, and R. Galán, "Building geometric feature based maps for indoor service robots," *Robotics and Autonomous Systems*, vol. 54, no. 7, pp. 546–558, 2006.

[10] T. Lozano-Pérez, "Spatial planning: A configuration space approach," *IEEE Trans. Computers*, vol. 32, no. 2, pp. 108–120, 1983.

[11] J.-C. Latombe, *Robot Motion Planning*. Kluwer academic publishers, 1996.

[12] G. Faria, R. Romero, E. Prestes, and M. Idiart, "Comparing harmonic functions and potential fields in the trajectory control of mobile robots," *Robotics, Automation and Mechatronics, 2004 IEEE Conference on*, vol. 2, pp. 762–767 vol.2, Dec. 2004.

[13] D. C. Conner, A. Rizzi, and H. Choset, "Composition of local potential functions for global robot control and navigation," in *Proceedings of 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)*, vol. 4. IEEE, October 2003, pp. 3546– 3551.

[14] E. W. Weisstein, ""saddle point." from mathworld–a wolfram web resource. http://mathworld.wolfram.com/saddlepoint.html."

[15] M. Trevisan, M. A. Idiart, E. Prestes, and P. M. Engel, "Exploratory navigation based on dynamical boundary value problems," *J. Intell. Robotics Syst.*, vol. 45, no. 2, pp. 101–114, 2006.

[16] E. P. e Silva Jr., P. M. Engel, M. Trevisan, and M. A. P. Idiart, "Exploration method using harmonic functions," *Robotics and Autonomous Systems*, vol. 40, no. 1, pp. 25–42, 2002.

[17] E. Kreyszig, *Advanced Engineering Mathematics*, 8th ed. John Wiley and Sons Inc, 2002.

[18] M. Karnik, B. Dasgupta, and V. Eswaran, "A comparative study of dirichlet and neumann conditions for path planning through harmonic functions," *Future Gener. Comput. Syst.*, vol. 20, no. 3, pp. 441–452, 2004.

[19] J. F. R. Rojas, *Neural Networks: A Systematic Introduction*. Springer-Verlag New York, Inc, 1996.

[20] C. M. Bishop, *Pattern recognition and machine learning*. Springer Science+Business Media, LLC, 2006.

[21] S. P. Singh, A. G. Barto, R. Grupen, and C. Connolly, "Robust reinforcement learning in motion planning," in *Advances in Neural Information Processing Systems 6*. Morgan Kaufmann, 1994, pp. 655–662.

[22] M. A. I. Fabio Dapper, Edson Pretes and L. P. Nedel, *Simulating Pedestrian Behavior with Potential Fields*. Springer Berlin / Heidelberg, 2006.

[23] R. W. Zurek and L. J. Martin, "Interannual variability of planet-encircling dust activity on Mars," vol. 98, no. E2, pp. 3247–3259, 1993.

[24] K.S.Narendra and K.Parthsarathy, "Identification and control of dynamical system using neural networks," *IEENN*, vol. 1, no. 1, pp. 4–27, 1990.

[25] S. J. Russell and P. Norvig, *Artificial Intelligence A Modern Approach*, 2nd ed. Pearson Education, Inc., 2003.

[26] J.-O. Kim and P. Khosla, "Real-time obstacle avoidance using harmonic potential functions," *IEEE Transactions on Robotics and Automation*, June 1992.

[27] Iniguez.P and Rosell.J, "Probabilistic harmonic-function-based method for robot motion planning," *IEEE/RSJ*, vol. 1, no. 1, pp. 382– 387, 2003.

BIOGRAPHICAL STATEMENT

Giles DSilva received his Bachelor of Science in Mathematics from Xaviers College, Mumbai, India 2003 and went on to complete his Master of Science in Information Technology from Chowgule College, Goa, India in 2005. An internship with Siemens Ltd and a short term contract with Tivim Technologies were followed by a Master in Computer Science from The University of Texas at Arlington, Texas in 2008. He has also interned with Sabre Inc. in Texas during his masters program and is currently contemplating pursuing a Doctoral program with Dr. Manfred Huber in the field of Robotics and Artificial Intelligence or working in a game development company.