LEARNING HIERARCHICAL TRAVERSABILITY REPRESENTATION FOR EFFICIENT MULTI-RESOLUTION PATH PLANNING

by

REZA ETEMADI IDGAHI

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

THE UNIVERSITY OF TEXAS AT ARLINGTON

May 2021

To my Sister,

who has always been the wind beneath my wings;

whose benevolence and kindness has always been the greatest enabler in my life.

ACKNOWLEDGEMENTS

I would like to thank Dr. Manfred Huber, my thesis advisor, for his invaluable guidance and support throughout my master thesis. I would also like to thank Dr. Farhad Kamangar and Dr. Deokgun Park, for accepting to be on my thesis committee.

I would sincerely like to thank my advisors at UTA whose support and advice is the reason I have been able to come this far: Ginger Dickens, and Sha'Londa Towns. I would like to thank my friends in Iran, Ahmad, Amin, and Mohammad for always looking out for me and motivating me throughout my master's program.

I am grateful to my parents and sister for all their unconditional love and support. For me to earn a master's degree was as much of their dream as it was mine.

April 29, 2021

ABSTRACT

LEARNING HIERARCHICAL TRAVERSABILITY REPRESENTATION FOR EFFICIENT MULTI-RESOLUTION PATH PLANNING

Reza Etemadi Idgahi, M.S.

The University of Texas at Arlington, 2021

Supervising Professor: Manfred Huber

Path finding on grid-based obstacle maps is an important and much studied problem with applications in robotics and autonomy. Traditionally, in the AI community, heuristic search methods (e.g. based on Dijkstra and A*, or based on random trees) are used to solve this problem. This search, however incurs significant computational cost that grows with the size and resolution of the obstacle grid and has to be mitigated with effective heuristics in order to allow path finding in real time. In this work we introduce a learning framework using deep neural networks with a stackable convolution kernel to establish a hierarchy of directional traversability representations with decreasing resolution that can serve as an efficient heuristic to guide a multi-resolution path planner. This path planner finds paths efficiently starting on the lowest resolution traversability representation and then refining the path incrementally through the hierarchy until it addresses the original obstacle constraints. We demonstrate the benefits and applicability of this approach on datasets of maps we created to represent both indoor and outdoor environments in order to represent different real world applications. The conducted experiments show that our method can improve the time of path planning by 30% in indoor environments and 66% in outdoor environments compared to the application of the same heuristic search method applied to the original obstacle map, which demonstrates the effectiveness of this method.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Path planning on a given static grid is one of the well-known and common problems of AI and Robotics which is solved with a large variety of methods that have been proposed so far. These methods can be divided broadly into 2 general categories. One is global path planners that compute a complete path before starting navigation, such as Dijkstra, A* or Rapidly Expanding Random Trees (RRT). The other is local path planners which determine local navigation actions without first computing a complete path, such as potential fields or reactive navigation strategies.

Local path planning generally has the advantage of requiring less computation time and of being able to work while only having knowledge about the local environment around the current position of the agent. However, these methods are prone to fail and get stuck in local minima and thus may not be able to find a path.

On the other hand, global path planners rely on heuristic search in the state-space induced by grid cells and can guarantee to find a path given sufficient information about the environment. However, since they require significantly more information compared to the local path planners, these methods often struggle to find a path in a reasonable amount of time as the size of the maps increase. While efficient heuristics can accelerate path construction in these algorithms, the derivation of effective heuristics for a specific environment is a significant challenge. Most traditional approaches thus use generic heuristics that are largely independent of the environment such as Cartesian Distance or Search Node Density, thus still requiring large amounts of time in cluttered and ill-structured environments. This is a huge drawback that makes these methods difficult to use for applications with stringent time constraints.

In this work we introduce a novel method to derive a hierarchical traversability representation using a stackable convolutional neural network architecture that can serve as an efficient heuristic for global path finding approaches. The proposed approach finds a path between two positions in a large map while reducing the time of path planning. To do this, we translate the obstacle grid map into a directional traversability representation and then learn a model in a supervised fashion that can reduce the resolution of traversability grid maps and create a new set of traversability maps of smaller size that preserves most navigation information. This model is created using a convolutional neural network that can be expanded without retraining to variable size inputs while achieving a constant factor size reduction. This enables us to to stack the learned convolutional kernel an arbitrary number of times without additional re-training, yielding an increasingly deep network and creating a growing hierarchy of abstracted maps of increasingly smaller size. On this hierarchy we then apply a global path planner in a multi-resolution framework, starting by finding a path in the smallest size (i.e. coarsest resolution) abstracted map and then incrementally refining it on the increasingly higher resolution maps on each level of the hierarchy, including the original grid map which results in the final path. The use of the traversability hierarchy here ensures that paths found on low resolution maps retain most aspects of the true path and can thus be easily refined, yielding faster overall planning times than planning on the full resolution map. While we use Dijkstra's algorithm here for path planning and refinement, the approach should also be applicable to other global planners, including A* and RRTs.

The main contribution of this work is to demonstrate that the model is able to learn a new, hierarchical representation of physical features in a map in terms of

obstacle and non obstacle cells. The learned model allows the algorithm to accurately reduce the resolution of the maps in order to perform path planning in less time. Furthermore, our experiments show that our algorithm is more efficient than the plain Dijkstra and can reduce the overall time of finding a path in a grid map significantly.

CHAPTER 2

RELATED WORK

A variety of search algorithms have been proposed in the context of path planning, such as A* [1], Anytime Repairing A* [2], Rapidly-exploring Random Tree (RRT) [3], Hybrid-A* [4], and two-stage RRT [5]. Generally these methods use generic, handcrafted heuristic functions, such as Manhattan distance, Euclidean distance, and length of Reeds-Shepp paths. Additionally in [6] it is shown that cost functions of the states can be created from the topology of the state and can be learned from demonstration [7]. Although it is possible to achieve optimal solutions with a well founded heuristic function, the complexity of the environments can make these methods unpractical specially when the length of the path or the dimension of the state space increases. ARA* [2] reduces the complexity by using a scaling factor and adjusting the upper bound of the cost of a path. Hybrid A* [4] uses the maximum of two different heuristic functions to guide grid cell expansion. However, it neither preserves completeness nor guarantees to achieve an optimal solution. Two-stage RRT [5] utilizes two RRTs where an upper RRT produces waypoints, which guide the other, lower RRT to address the robot kinematics.

Researchers have developed several hierarchical path planning methods in order to improve the quality of the planned path in complex environments [8, 9, 10, 11, 12]. Fujimura and Samet [11] proposed a hierarchical system for path planning in an environment with moving obstacles, in which time was included as one of the dimensions of the model world. However, this approach leads to having a large search space. In [12], a hierarchical approximate cell decomposition method was introduced for path planning where different resolutions were used in cell decomposition. Additionally path planning methods based on fuzzy systems and optimization methods have been introduced by researchers in order to overcome the challenging properties of environments such as unknown and dynamic areas [8, 13]. In [13], a multi-objective path planning algorithm based on particle swarm optimization is proposed for robot navigation in uncertain environments. By contrast, [8] proposed a hierarchical planning strategy with two layers, where fuzzy logic was used for the proposed motion planning strategy. However, the optimization and generalization ability of these planners still need to be improved.

Fueled by the increasing popularity of artificial neural networks, the AI community has started to utilize these computational models in path planning and use their powerful features such as strong function approximation. In particular, deep reinforcement learning [14] which is a machine learning framework for sequential decision making problems has been used to learn a policy that can generate the shortest path. Prior RL algorithms, like Q-learning and Sarsa [14], work well with discrete state spaces. Konar et al. [15] have applied Q-learning to path planning of a mobile robot. However, if the state spaces become very large or continuous, these algorithms will be computationally expensive and impractical for applications. In [16, 17, 18, 19] researchers introduced various approximation methods to deal with large or continuous state spaces. Inspired by the least-squares temporal difference learning algorithm [16], Lagoudakis and Parr [17] proposed the least-squares policy iteration algorithm (LSPI), in which linear architectures were used to approximate the value functions in continuous state spaces. In [20] a hierarchical planning approach based on A* and LSPI is presented in which a two-level structure was used. In the first level, the A* algorithm was used to find several path points as subgoals for the next level and in the second level, LSPI was used to learn a near-optimal local planning policy. However, these methods require significant training in order to be practical for real world applications.

In contrast to this previous work, the approach presented here uses deep neural networks to explicitly learn reductions of the world model that can then be used as a heuristic by a path planner to reduce complexity. The goal here is to combine the benefits of a global, well studied path planner in terms of completeness and correctness with the advantages of machine learning to identify patterns in traversability representations that indicate successful path attributes.

CHAPTER 3

METHOD

The method developed here attempts to provide efficient guidance to a hierarchical path planning and refinement approach by constructing a cascade of increasingly lower resolution representations of the environment that maintain the main characteristics of the environment in terms of path construction with minimal refinement requirements. For this, it is important that the maps maintain broad traversability characteristics rather than obstacle locations or densities. For example, while an orchard of trees and a field with multiple walls might have the same average obstacle density, traversability of the orchard is relatively universal (i.e. there is a path through the orchard in pretty much any direction), traversability of the walled field is severely limited. When abstracting these two pieces of environment in a way that preserves approximate paths, it is thus important that these environments are represented so their characteristics are preserved. For this, we propose here the concept of directional traversability maps to capture this information and utilize supervised learning with convolutional networks to identify the features that indicate particular traversability characteristics.

To be able to address varying size maps, the developed learning approach builds a convolutional kernel in the network that reduces the traversability representation of a map region into a lower resolution traversability representation of the same region where the resolution reduction is a constant factor. This allows us not only to expand the convolutional layer to arbitrary input sizes but also allows the same kernel to be stacked in additional layers of a deeper network, producing increasingly coarser resolution traversability maps with each additional stacked layer.

Using this hierarchy of representations, path planning then works back up the hierarchy, starting by constructing a path in the lowest resolution (and thus smallest size) map and then incrementally refining it by considering the next higher level representation.

Figure 3.1 shows an overview of the heuristic construction and path planning approach.

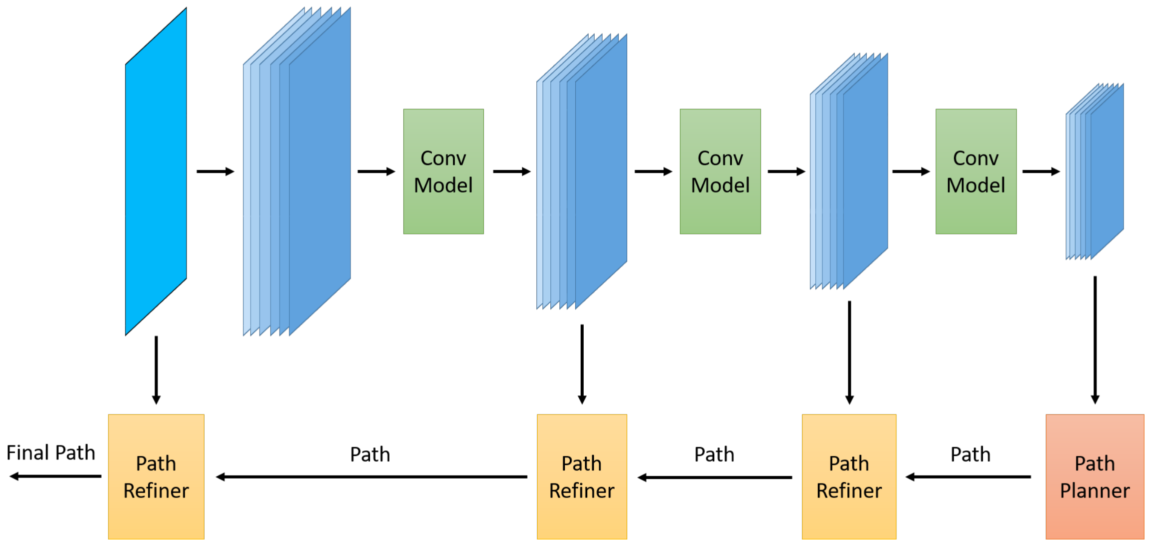

Figure 3.1. Overview of the proposed hierarchical, multi-resolution path planning approach. Construction of the reducing resolution directional traversability cascade (top from left to right) is followed by bottom up path planning and refinement (bottom from right to left).

Starting by translating the original obstacle map (top left), into a set of directional traversability maps, these are then transformed into incrementally reduced size representations through the application of the learned convolutional network kernel until a desired smallest size is reached (illustrated left to right at the top of the figure). Once this representation hierarchy is formed, the path planner is executed at the bottom of the hierarchy (bottom right of the figure) and the results is incrementally refined moving back up the hierarchy until a path is derived on the original obstacle map (illustrated right to left at the bottom of the figure).

3.1 Directional Traversability Maps

In this work, the gird maps that represent the environment are called obstacle maps. In these maps each cell can be either 0 meaning obstacle or 1 meaning non-obstacle cell. We introduce the concept of directional traversability maps as a new way of representing obstacle maps where each cell can have a value between 0 and 1 which is called its traversability score, loosely representing the likelihood that the represented region is traversable in the intended direction. Each obstacle map can be converted into multiple traversability maps where each map shows the traversability scores in a specific direction. In the case of 6 directions, these directions are horizontal, vertical, top-left, top-right, bottom-left, and bottom-right, where the latter names indicate the entry and exit edges of the traversal relative to the considered region. The traversability score for each direction represent the degree to which a path exists that traverses the underlying patch of the environment in the corresponding general direction. Figure 3.2 shows an example of an obstacle map region and the corresponding traversability representations.

To manually convert an obstacle map to a set of equal sized traversability maps, we need to calculate the traversability scores for each cell in the obstacle map. If the cell is an obstacle then traversability scores in all directions are set to 0. Otherwise if the cell is non-obstacle, we look at the pair of adjacent neighbors of that cell in each direction. For example, the left and right neighbors are considered for horizontal, and top and bottom for vertical traversability. The traversability score in each direction

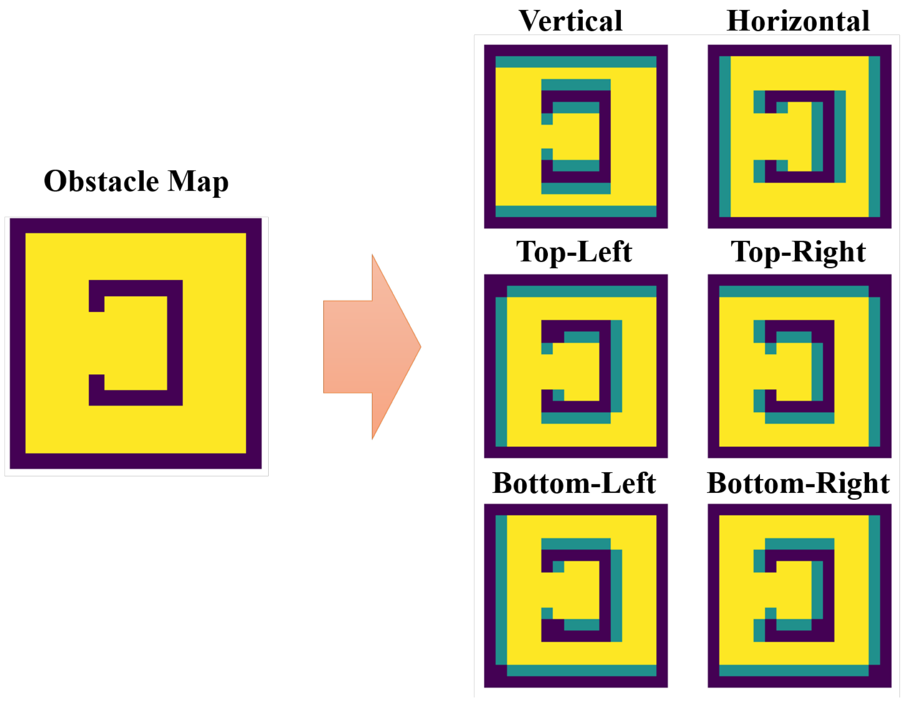

Figure 3.2. Obstacle map and corresponding directional traversability representations for the 6 traversal directions.

is set to 0 if both neighbors are obstacles, 1 if both are non-obstacles, and 0.5 if one is an obstacle and the other is a non-obstacle.

Generating general traversability maps at lower resolutions where each location in the map represents an entire region of space is more complex. In order to do this manually, a search process would be required to explore all possible ways to traverse the underlying region in the intended direction and to determine the likelihood that a particular entry and exit location pair in the specific direction would have a successful path through the underlying region. The complexity of this computation, however, increases significantly with the size of the region for which traversability is to be calculated, making such manual calculation of traversability unwieldy in large environments. To address this, the work presented here trains a convolutional neural network to predict traversability scores for larger regions hierarchically from traversability scores from smaller, underlying regions.

3.2 Model for Map Size Reduction

At the core of the proposed work is the construction of a cascade of resolution (and thus size) reduced directional traversability maps representing the same environment. To achieve this, we trained a convolutional model with 2 layers, with the goal of the learned convolutional kernel to reduce a 4x4 patch of the 6 directional traversability maps to a 2x2 reduced size patch of 6 more abstract directional traversability scores. The used convolutional kernel, when applied in a convolutional layer, thus reduces the resolution of the traversability maps by a factor of 2 in each dimension. In the used network, the first layer of the convolutional kernel has 3 4x4 filters with a stride of 2 and relu activation functions for each directional traversability map. The outputs of this first layer of the kernel are stacked and as a result the input traversability maps with size of WxHx6 are converted to W/2xH/2x18 maps which serve as the input for the second layer of the convolution kernel. The second layer is a 3D convolutional layer that has 6 1x1x18 filters with a stride of 1 and clipped relu activation functions with max value of 1. This function enables the model to generate valid values between 0 and 1 as traversability score. Therefore, the output of this models is 6 directional traversability maps with half resolution of the input traversability maps. The convolution kernel can then be replicated in a variable size convolution layer that covers the entire size of the presented map, making it flexible to train and apply on arbitrarily large maps. Figure 3.3 shows an illustration of the proposed model.

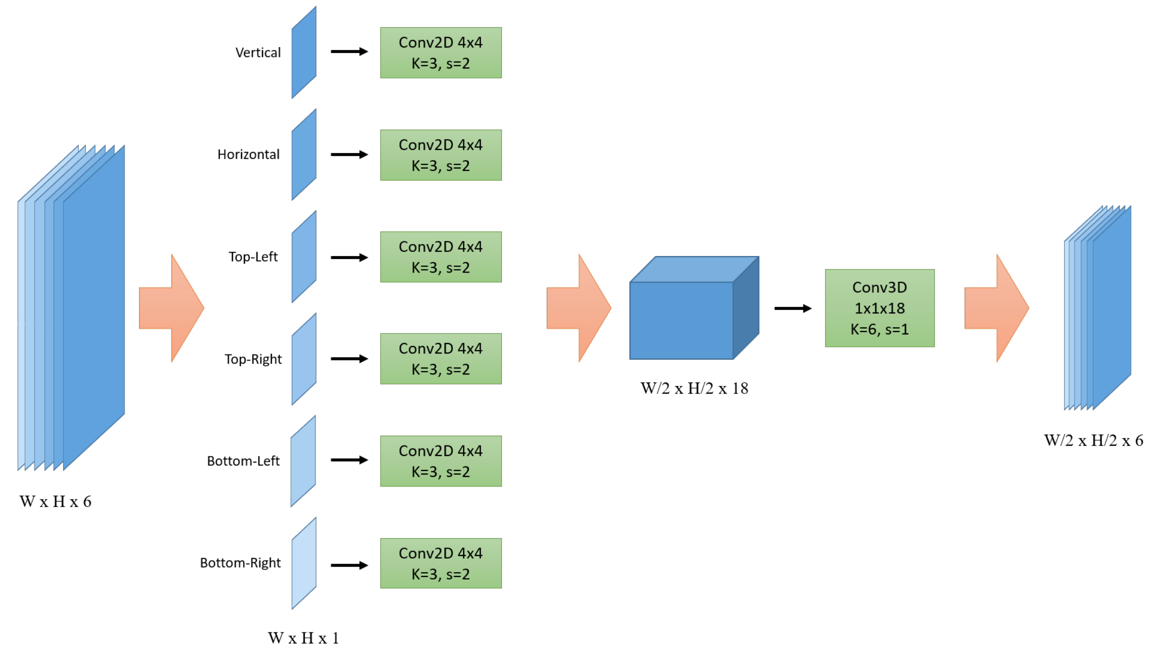

Figure 3.3. Proposed model of convolutional kernel for map size reduction.

3.2.1 Training Data Generation

To train the convolutional network for one step of size reduction, we generated a training set of obstacle maps with corresponding resolution reduced traversability labels using an automated labeling process based on a local path search in the underlying region. The label is defined as directional traversability maps with half resolution created for each training obstacle map. In the case of the maps used for our experiments in Chapter 4, for example, where our original obstacle and traversability maps are 256x256 maps, these label maps have a size of 128x128 and each cell on these maps represent a 4x4 window on the obstacle map in the form of a traversability score. This score is a real number between 0 and 1 where 0 means there is no path in the specified direction in that window and 1 means there are paths from any point on one side of the window to any point on the other side in the specified direction. Values in between represent the fraction of location pairs on the entrance and exit side of the area corresponding to the given direction that can be connected through the area with a collision free path. Loosely, this corresponds to a measure indicating how easy it would be to find a path traversing the area in the given direction. The window moves on the obstacle map with a step size of 2 which creates overlapping areas in the form of 2 shared columns or rows between two adjacent windows. This helps to have consistent traversability maps and catch all the different formations of obstacles between 2 adjacent windows.

It is important to note here that this training data only contains traversability scores of 0, 0.5, and 1 on the input side while the final learned reduction network will have to be able to be applied to arbitrary traversability scores in order to allow for the hierarchical cascade with multiple size reduction steps. The goal here is for the generalization ability of the neural network to translate the examples in this limited training set into a more general transformation, thus alleviating the need for constructing larger scale training data where automatic labeling would be substantially more computationally expensive.

3.2.2 Hierarchical Scale Reduction

Using the derived training data, we train a convolutional network with one layer of the two layer convolutional kernels. The result of this is a trained convolutional kernel that can efficiently derive a traversability representation of the same area but with half the resolution in both width and height. Since we are using convolutional layers in this model, the size of the input can be changed dynamically by increasing or decreasing the number of convolutional kernels in the layer. This, together with the fact that the convolutional kernel utilizes the same input and output representation in terms of 6 traversability values, enables us to stack multiple layers with identical convolution kernels in order to achieve an incremental reduction in size down to a desired target size. For this, we can train the model once and use it for all the levels of abstraction, minimizing the training required while obtaining an approach that is applicable to arbitrary size maps. The concept of this multi-layer application is shown as an overview in Figure 3.1 and on an example obstacle map in Figure 3.4.

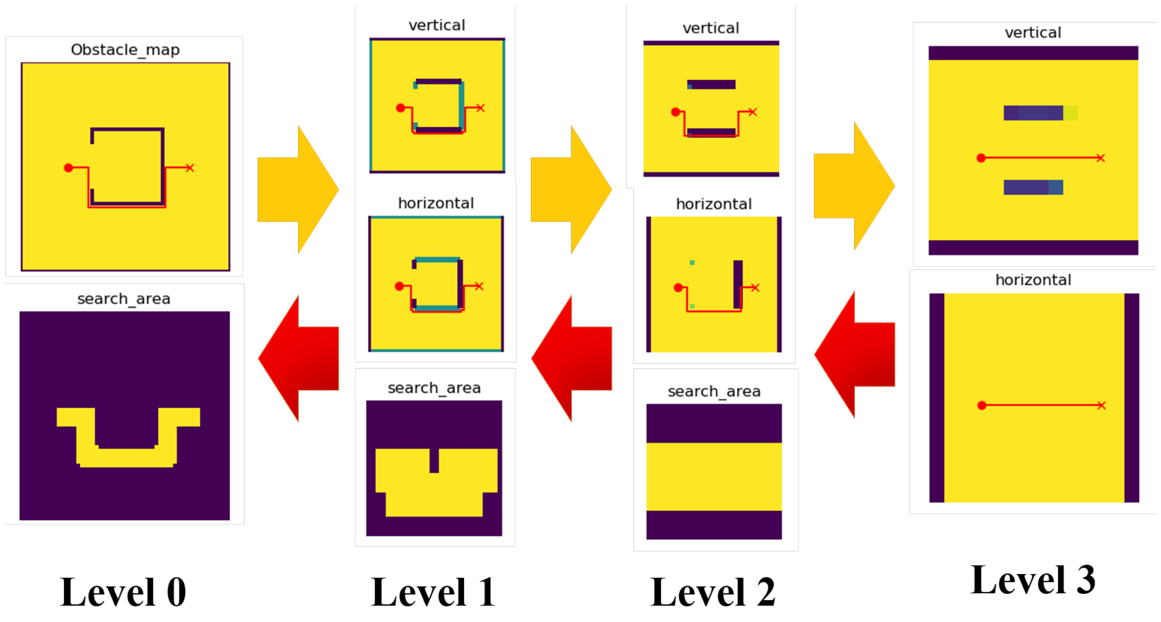

Figure 3.4. Example of the hierarchical path planning approach using 3 abstraction levels. Hierarchical scale reduction of directional traversability maps (left to right) with corresponding path construction and refinement (right to left). Traversability scores in the horizontal and vertical traversability maps in the top two rows are indicated by level of shading and the refined path at each level is shown in red. Search area maps in the bottom row show the area around the path used for path refinement.

3.3 Hierarchical Path Planning

After creating the dataset, the supervised learning approach is used to train a model that can convert traversability maps to lower resolution maps while maintaining the important features for path planning. This model then enables us to use it on a map multiple times in a hierarchy in order to reach a very small map where a global path planner can perform reasonably well. In this thesis, Dijkstra's algorithm is used to find this path but a different path planner, such as RRT, could also be used. After a path at the lowest resolution has been found, the goal now is to incrementally refine it in the context of the higher resolution traversability maps until a path in the original environment is found. To achieve this, it is important to have a version of the path planner that can perform local refinement of a path. For that, a modified version of Dijkstra is used here to find a path in the derived map by using the traversability maps at the current level and a local search area specified around the path derived on the previous, more abstract level. Using this, the path is refined in each level where increasingly more details of the map are available. This is repeated until the original map is reached.

While the local refinement with its limited search region can dramatically reduce the computation time required, there is the potential that it is not complete and that it therefore might sometimes fail to refine a a path within the limited search region. To address this, in situations where the path refinement fails to find a path at any given level, the path planner is re-run on the complete map at this level to provide a new seed path for the remaining abstraction levels. While this can lead to an increase in the total planning time, it ensures that the completeness and correctness properties of the underlying path planning approach are preserved.

3.3.1 Modified Dijkstra

For path planning algorithm, Dijkstra is used here with a few modifications that allow us to use it on traversability maps. The input of this algorithm are 6 directional traversability maps with values between 0 and 1. To cope with this multidimensional input, the following heuristic function is introduced in order to include the traversability scores:

$$c_{dir} = d + \lambda(1 - t_{dir})$$

where c_{dir} is the cost of the current cell with a traversal in direction dir, d is the length of the current shortest path from the starting point to the entry of the current cell, t_{dir} is the traversability score of the current cell in this direction, and λ is a hyper-parameter that specifies the importance of the traversability score compared to the length of the path. Note that if the traversability score is 0, that cell will not be considered for path planning because it is either unreachable or untraversable, meaning we can reach it from one side but we can not exit from another side. This rule has an exception at the start and end point of the path where we just want to reach these cells and not traverse them.

Algorithm 1 presents a coarse algorithm description of the overall approach to hierarchical traversability map-based multi-resolution path planning and Figure 3.4 shows and example of the map cascade and hierarchical path refinement on an example world.

Figure 3.5 and 3.6 show an example of hierarchical abstraction and path refinement respectively on a full size, 256x256 resolution map with 6 traversability directions and three levels of abstraction.

Algorithm 1 Hierarchical Path Planning

```
1: function HPP($obstacleMap, maxLevel$)
2:     Produce traverse map from obstacle map
3:     Append traverse map to traverse list
4:     for $level \leftarrow 1$ to $maxLevel$ do
5:         $traverseMap \leftarrow$
6:             $convModel(traveseList[level - 1])$
7:         Append traverse map to traverse list
8:     $path \leftarrow pathPlanner(traverseList[maxLevel])$
9:     for $i \leftarrow maxLevels - 1$ to 1 do
10:        $path \leftarrow pathRefiner(path, traverseList[i])$
11:    $path \leftarrow pathRefiner(path, obstacleMap)$
12:    return $path$
```

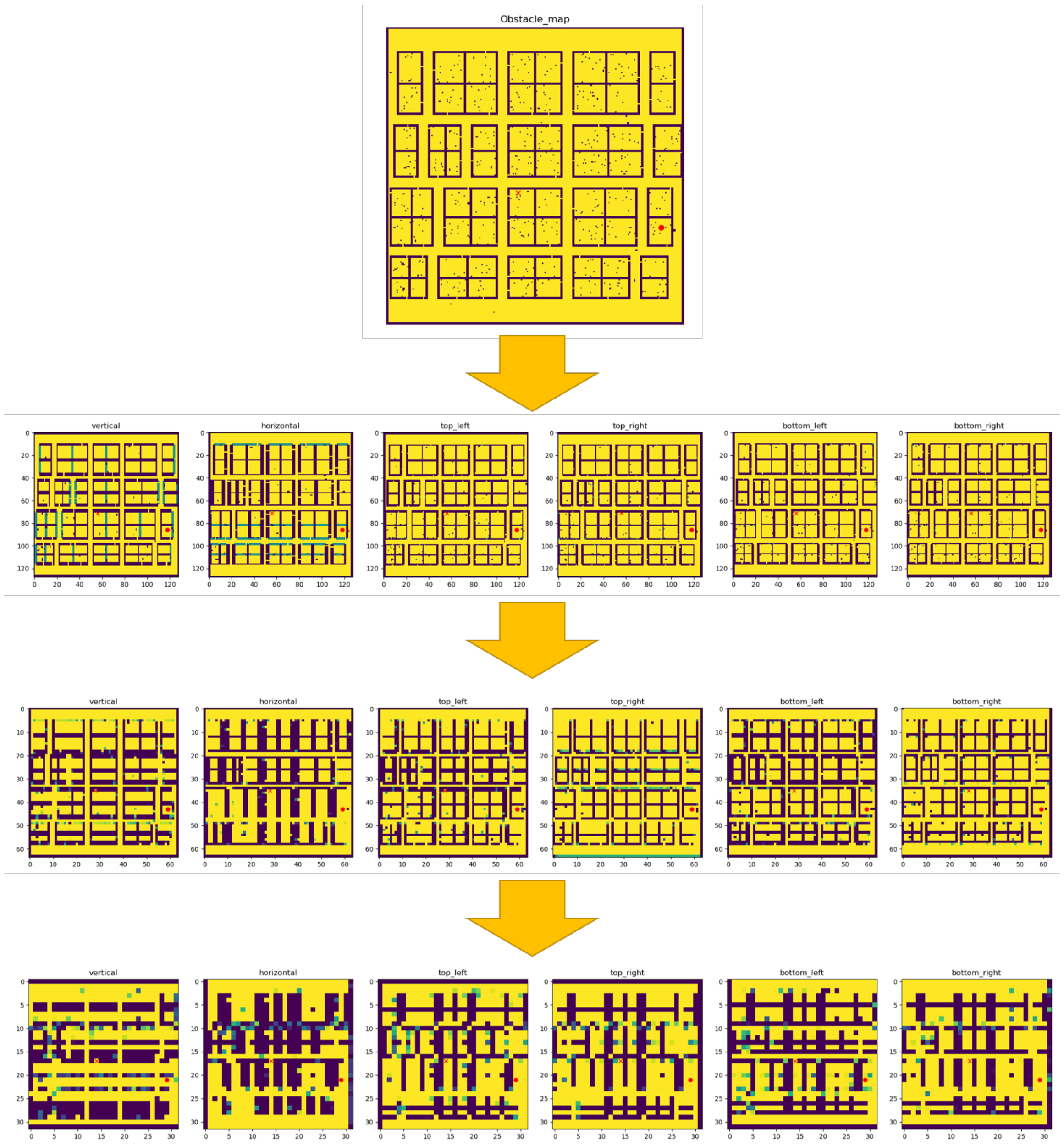

Figure 3.5. Hierarchical abstraction on a full size, 256x256 resolution map with 6 traversability directions and three levels of abstraction from top to bottom.

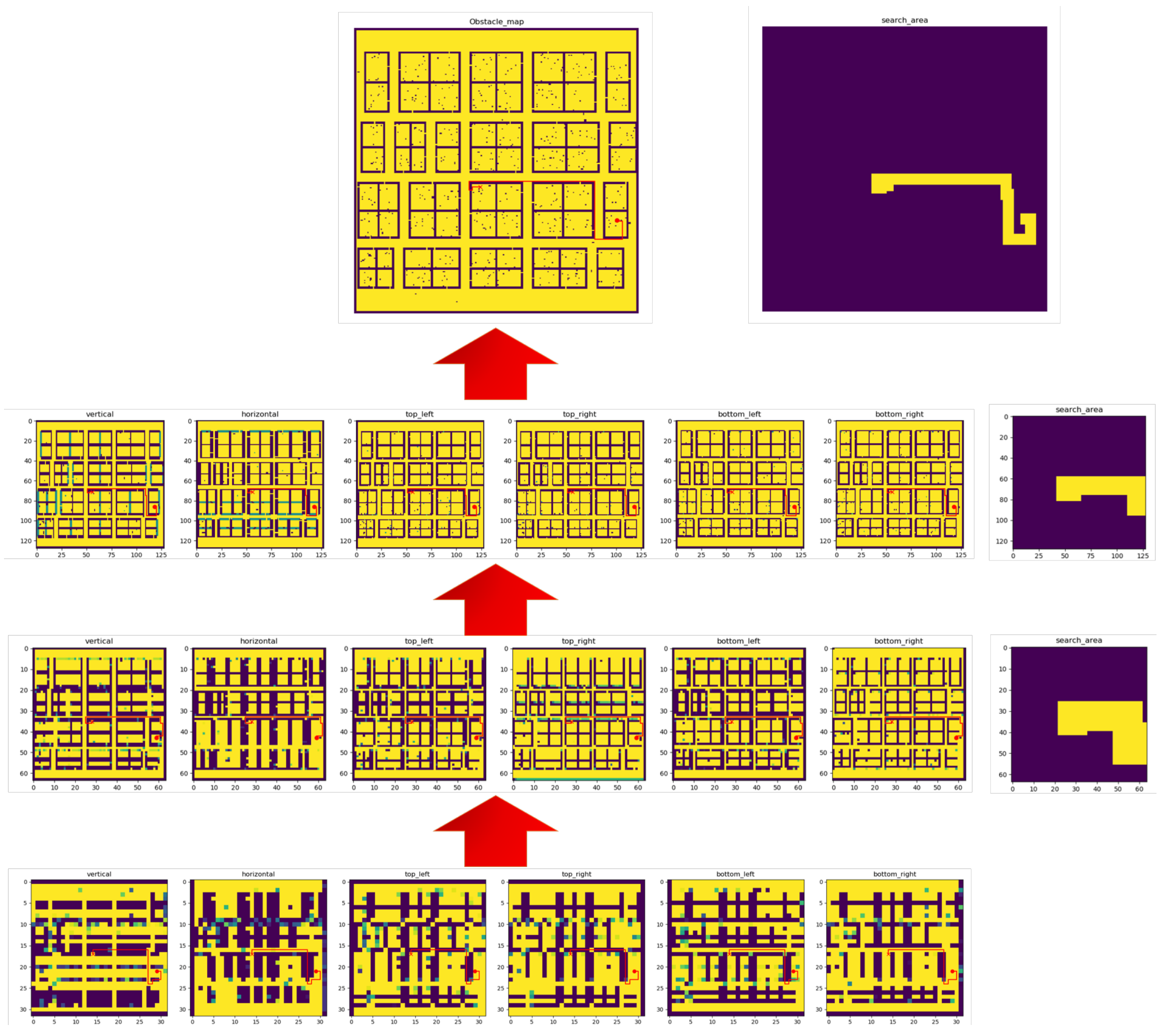

Figure 3.6. Path refinement on a hierarchy of maps with three levels of abstraction from bottom to top.

CHAPTER 4

RESULTS

In the experimental section, we would like to answer the following questions: (1) How does this method perform in terms of the time required to find a path, compared to running raw Dijkstra? (2) How does the trained model generalize and represent environments accurately as the levels of abstraction goes further? (3) How often does the path refiner fail to refine a path in a map with more details and how much time is wasted on these maps as the planner has to replan a path at the failing level rather than just refine the existing path ? (4) What is the relation between the number of reduction levels and the saved time in path planning?

To answer these questions, we evaluate our method on 2000 indoor maps and 2000 outdoor maps with random start and end positions.

4.1 Dataset

To evaluate the approach in a variety of settings and to derive training data for the learning approach to hierarchical resolution reduction of the directional traversability maps, an environment generator was constructed that constructs arbitrary worlds with particular characteristics representing indoor and outdoor environments. With this, we created 2 sets of 256x256 resolution maps to represent both indoor and outdoor environments. Indoor maps consist of rooms and hallways with few random obstacles which represent furniture or other obstacles that may be seen in a building. By contrast, outdoor maps consist of forest, field, and jungle patches which are nothing but large numbers of small obstacles, such as trees scattered through an area with different densities. There are also random walls and obstacles of larger size in these maps. During the creation of the datasets, parameters such as size of hallways and number of random obstacles for indoor maps, and parameters including size and density of the jungles, as well as position and size of the walls and obstacles for outdoor environments were specified randomly in order to create a diverse dataset that can represent many kinds of environments.

4.1.1 Creating Outdoor Maps

As mentioned, outdoor maps are made of jungle patches, random walls, and obstacles. In the created dataset, outdoor maps are made of 8 jungle patches with random position, width, height, and density. The width and height of the patches are constrained between 50 to 130 pixels, and density cannot exceed 0.25 which means at most 25% of pixels in a patch can be obstacles. While having randomness in choosing the properties of patches helps to represent the unstructured nature of outdoor environments, adding these constraints helps to control the size of jungles and leave room for other features such as empty areas, walls, and random obstacles. In the next step, 5 walls are added to the map with random directions (can be horizontal or vertical) and random thickness (can be 1, 2, or 3) and a random length between 10 and 80. At the end, 40 random obstacles with arbitrary width and height up to 8 pixels are added to the map. Figure 4.1 shows a few examples of generated outdoor environments.

4.1.2 Creating Indoor Maps

Unlike outdoor environments, indoor maps have an organized structure formed by rooms and hallways. However there are different characteristics such as size and position of these elements that make indoor maps different from each other. In order

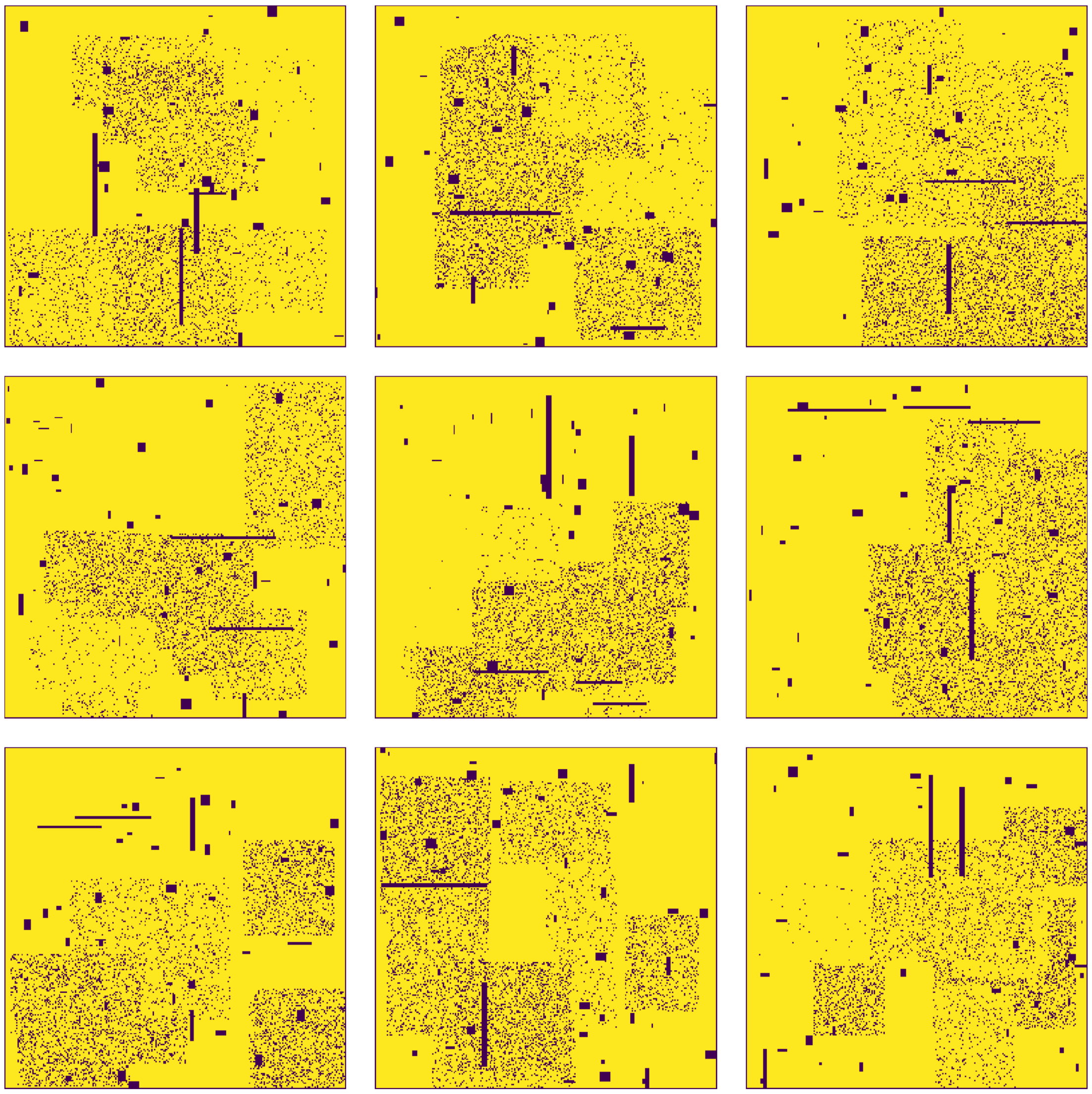

Figure 4.1. Examples of obstacle maps for outdoor environments.

to create a random indoor map with an underlying structured formation, a special algorithm is introduced which utilizes a randomly created block to construct the whole map. The indoor maps are formed by blocks which are simple rectangles that will contain rooms, and the space between blocks form the hallways in the map. After creating a block with random position and size in the map, for each edge of rectangle, as long as there is free space between that edge and the border of the map in that direction, a new block with the same size on one edge and random size on the other edge is created. For example the new blocks on the left and right side of the first block have the same height but random width, and on top and bottom, they have the same width but random height. In other words, after creating the first block, new blocks are created in four directions in the form of a plus sign, representing rooms along major hallways. After that, the same routine is repeated to create new blocks around the previously created blocks until there is no more large enough free area in the map. Finally, each block is divided into 1, 2, or 4 rooms based on the size of the block and doors are added for each room. Figure 4.2 shows the process of creating an indoor map, and Figure 4.3 shows a few examples of generated indoor maps.

4.2 Performance Evaluation

To answer the above-mentioned questions and evaluate the different performance characteristics of the proposed hierarchical abstraction and path planning approach, we run Dijkstra as well as our method on each of the 2000 outdoor and 2000 indoor maps and measure the time each algorithm consumes to find a valid path. Due to the existence of uncontrollable conditions in CPU processing time, we run each algorithm 4 times and report the average of the measured times for better accuracy.

Additionally, we tested our method with different numbers of reduction levels in order to measure the saved time as the levels are increased. Table 4.1 demonstrates that our method with 3 levels of reduction can improve the time of path planning by 30% on indoor environments and 66% on outdoor environments.

However, due to the fact that at each level of resolution reduction some of the details are removed, there are some cases in our experiment where at some level the path refiner fails to refine the given path on the map with more details. In these cases,

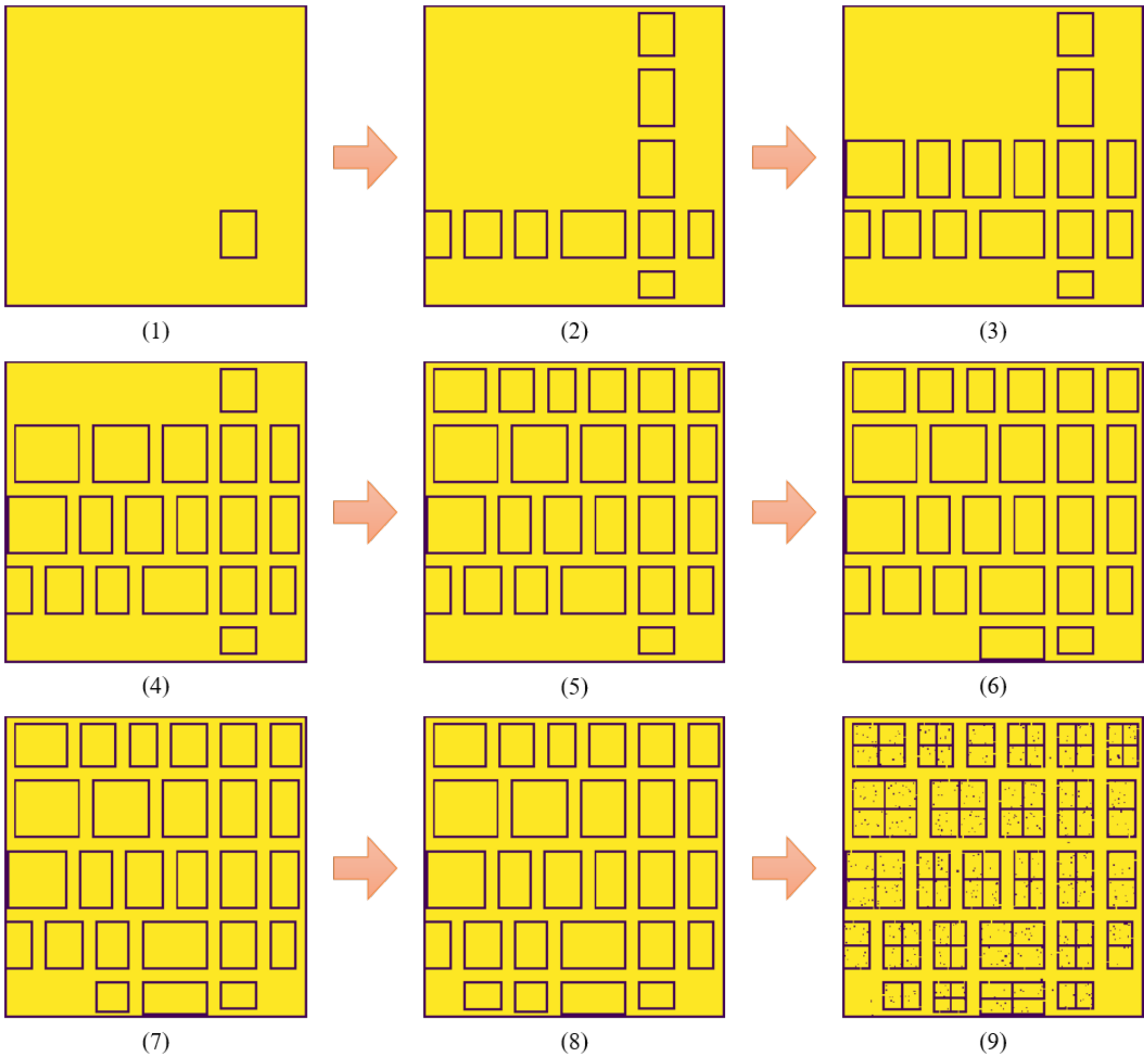

Figure 4.2. The process of creating an indoor map.

the path planner finds a path on the lowest resolution map, however this path is not feasible because the map is abstracted multiple times and some important features for path planning may be lost during the process. If the path refinement fails, the path planner has to plan a new path at the failing level, thus wasting time and potentially taking longer than the base path planner would. While the data in Table 4.1 includes both the cases with and without failures, it would be beneficial to analyze the cost

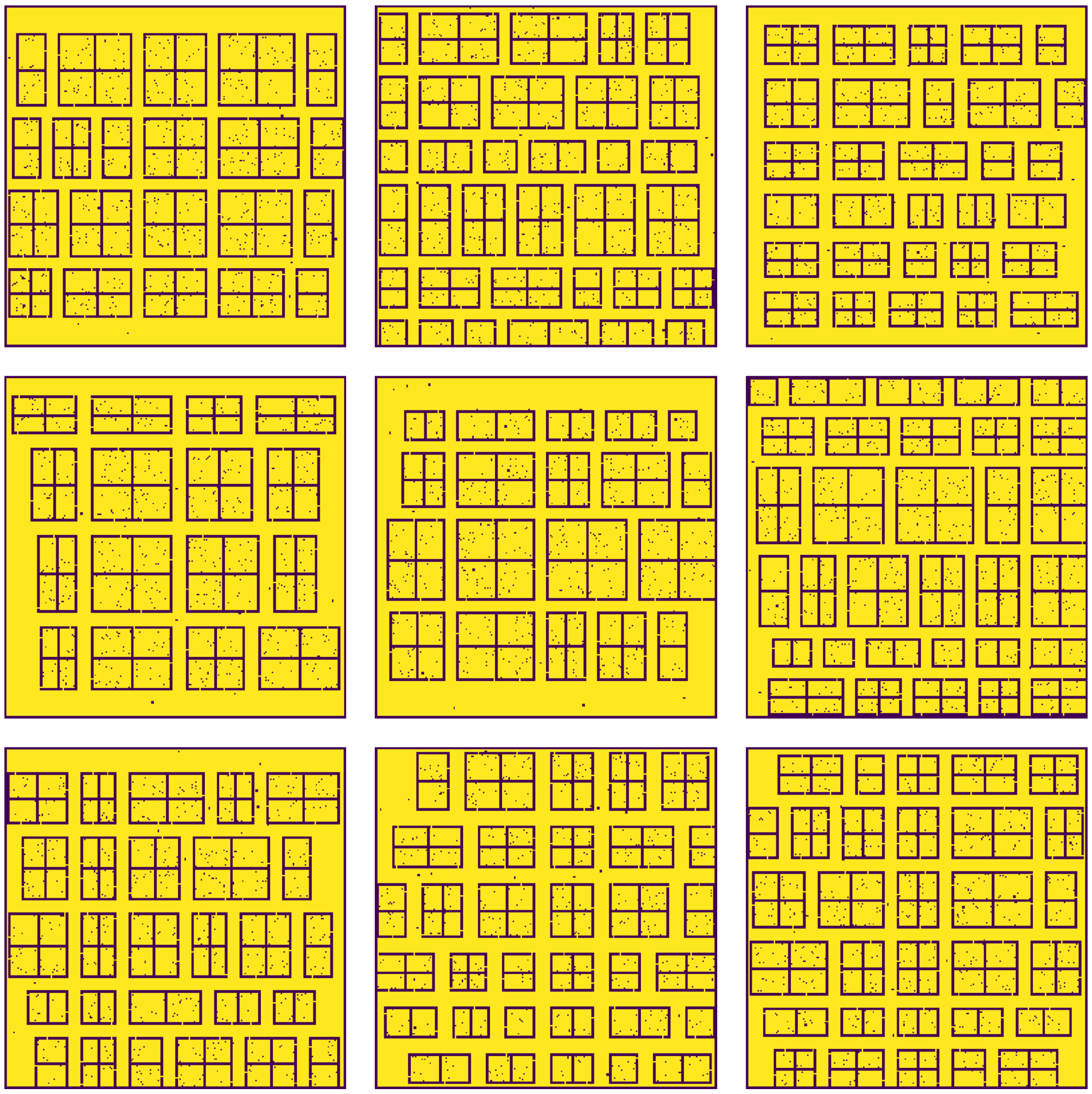
Figure 4.3. Examples of obstacle maps for indoor environments.

of failures at different levels in order to obtain a better estimate of the expected best and worst case speedups and slowdowns that can occur.

Table 4.2 shows the number of failures of the path refiner at each level as well as the wasted time. The wasted time is the time that our method has spent on finding a path and refining it before reaching the level at which the failure happens. That

Table 4.1. Average time of path planning

	Indoor	**Outdoor**
Raw Dijkstra	1.244451	1.606247
HPP 1 Level	0.92951	0.762387
HPP 2 Levels	0.802333	0.56627
HPP 3 Levels	0.758200	0.523608

path is not feasible and we have to run the path planner to search the whole map and find a new path at that level. After that the normal operation continues and the new path is given to the levels after for refinement.

Table 4.2. Number of failures and time loss at each level among the 2000 environments for both indoor and outdoor environments.

	Indoor		**Outdoor**	
Failure Level	Failures	Wasted Time	Failures	Wasted Time
Level 0	36	0.36504	0	0
Level 1	186	0.07783	11	0.08055
Level 2	102	0.02024	16	0.02302

From Table 4.2, it can be seen that this phenomenon is a more significant effect in the indoor environments as the number of failures is higher. The reason is that in these maps doors are small features that provide access between hallways and rooms and play a major role in the formation of the path. Combining this with random obstacles around these areas can create challenging situations that our model may not be able to represent accurately on the reduced sized map and therefore leads to finding infeasible paths in abstracted maps.

However, the number of failures is still relatively small and the amount of excess time needed to correct, especially for failures at lower levels, is relatively limited, still resulting in competitive performance even in failure cases and significant gains on average as indicated in Table 4.1.

CHAPTER 5

DISCUSSION AND CONCLUSIONS

We have presented a novel method for path planning in a hierarchical manner. We introduced the concept of directional traversability maps and how we can represent an obstacle map in the form of directional traversability scores. To create the hierarchy, a convolutional model was trained using supervised learning to learn the traversability function as a convolutional network kernel and reduce the resolution of its input. This convolutional kernel can then be replicated into convolutional layers and stacked in a deep network that can derive a cascade of incrementally resolution reduced traversability representations of the environment. After that a modified version of Dijkstra was introduced to find and refine a path throughout the hierarchy and return the final path on the input obstacle map.

We demonstrated the potential for complexity reduction of our method in experimental results in both indoor and outdoor environments and showed that our method outperforms the raw Dijkstra and improves the time of path planning significantly on both types of environments. Additionally, we discussed the drawback of our method where some of the details of the maps are omitted during resolution reduction which leads to finding a path that is infeasible on higher resolution maps.

Regarding future work, one interesting direction is instead of using the precoded concept of traversability and supervised learning, to train the model using reinforcement learning and let it to come up with its own representation of an abstracted map which can be more useful for the path planner and further improve the performance while reducing the number of failures in the path refiner by better incorporating the important features in abstracted maps. Moreover, we are planning to utilize the hierarchical traversability heuristic in the context of an RRT based path planner to further study its potential in larger environments.

REFERENCES

[1] P. E. Hart, N. J. Nilsson, and B. Raphael, “A formal basis for the heuristic determination of minimum cost paths,” *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[2] M. Likhachev, G. J. Gordon, and S. Thrun, “Ara*: Anytime a* with provable bounds on sub-optimality,” *Advances in neural information processing systems*, vol. 16, pp. 767–774, 2003.

[3] S. M. LaValle *et al.*, “Rapidly-exploring random trees: A new tool for path planning,” 1998.

[4] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, “Practical search techniques in path planning for autonomous driving,” in *Proceedings of the First International Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08)*, 2008.

[5] Y. Wang, D. K. Jha, and Y. Akemi, “A two-stage rrt path planner for automated parking,” in *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, 2017, pp. 496–502.

[6] S. Mahadevan and M. Maggioni, “Proto-value functions: A laplacian framework for learning representation and control in markov decision processes.” *Journal of Machine Learning Research*, vol. 8, no. 10, 2007.

[7] N. D. Ratliff, J. A. Bagnell, and M. A. Zinkevich, “Maximum margin planning,” in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 729–736.

[8] X. Yang, M. Moallem, and R. Patel, “A layered goal-oriented fuzzy motion planning strategy for mobile robot navigation,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 35, no. 6, pp. 1214–1224, 2005.

[9] X.-C. Lai, S. S. Ge, and A. A. Mamun, “Hierarchical incremental path planning and situation-dependent optimized dynamic motion planning considering accelerations,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 6, pp. 1541–1554, 2007.

[10] D. Dolgov, M. Montemerlo, and J. Diebel, “Path planning for autonomous vehicles in unknown semi-structured environments,” *I. J. Robotic Res.*, vol. 29, pp. 485–501, 04 2010.

[11] K. Fujimura and H. Samet, “A hierarchical strategy for path planning among moving obstacles (mobile robot),” *IEEE Transactions on Robotics and Automation*, vol. 5, no. 1, pp. 61–69, 1989.

[12] D. Zhu and J.-C. Latombe, “New heuristic algorithms for efficient hierarchical path planning,” *IEEE Transactions on Robotics and Automation*, vol. 7, no. 1, pp. 9–20, 1991.

[13] Y. Zhang, D.-W. Gong, and J.-H. Zhang, “Robot path planning in uncertain environment using multi-objective particle swarm optimization,” *Neurocomput.*, vol. 103, p. 172–185, Mar. 2013. [Online]. Available: https://doi.org/10.1016/j.neucom.2012.09.019

[14] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998.

[15] A. Konar, I. Goswami Chakraborty, S. J. Singh, L. C. Jain, and A. K. Nagar, “A deterministic improved q-learning for path planning of a mobile robot,” *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 43, no. 5, pp. 1141–1153, 2013.

[16] S. J. Bradtke and A. G. Barto, "Linear least-squares algorithms for temporal difference learning," *Machine learning*, vol. 22, no. 1, pp. 33–57, 1996.

[17] M. G. Lagoudakis and R. Parr, "Least-squares policy iteration," *The Journal of Machine Learning Research*, vol. 4, pp. 1107–1149, 2003.

[18] X. Xu, C. Liu, S. X. Yang, and D. Hu, "Hierarchical approximate policy iteration with binary-tree state space decomposition," *IEEE Transactions on Neural Networks*, vol. 22, no. 12, pp. 1863–1877, 2011.

[19] X. Xu, Z. Hou, C. Lian, and H. He, "Online learning control using adaptive critic designs with sparse kernel machines," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 5, pp. 762–775, 2013.

[20] L. Zuo, Q. Guo, X. Xu, and H. Fu, "A hierarchical path planning approach based on a$*$ and least-squares policy iteration for mobile robots," *Neurocomputing*, vol. 170, pp. 257–266, 2015.

BIOGRAPHICAL STATEMENT

Reza Etemadi Idgahi was born in Mashhad, Iran, in 1996. He received his Bachelor of Engineering degree from Ferdowsi University of Mashhad, Mashhad, Iran, in 2018, in Computer Engineering, his Master of Science from The University of Texas at Arlington in 2021, in Computer Science under the supervision of Dr Manfred Huber. His research topic mainly focuses on the applications of machine learning in robotics and autonomy and discovering new methods to optimize traditional solutions for common problems in autonomy.