DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Biologically Inspired Navigation System in Dynamic Environments

Haoyang Sun

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Biologically Inspired Navigation System in Dynamic Environments

Navigations System mit einem biologischen Modell in dynamische Umgebungen

Author:	Haoyang Sun
Supervisor:	Prof. Dr.-Ing. habil. Alois Christian Knoll
Advisor:	Dr. rer. nat. Zhenshan Bing
Submission Date:	15.04.2023

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.04.2023 Haoyang Sun

Abstract

Different types of brain cells, including grid cells, place cells, and prefrontal cortex cells, allow rodents to build a cognitive map of their environment. Our Bio-Inspired Navigation (BioNav) method uses a two-phase approach to enable agents to navigate through an environment. During the exploration phase, the agent builds a cognitive map, and during the navigation phase, the agent uses the map to reach its goal. Our method offers two types of navigation: vector-based and topology-based. While the previous work on this method was effective, it lacked flexibility to dynamic changes in the environment and was prone to errors when the settings of exploration changed. Our work addresses these shortcomings and demonstrates the superiority of BioNav over conventional methods, such as D* Lite, through comparison experiments. We achieve this by implementing a record of visited places to provide more flexibility to changes in the environment and by using a simpler coordinate system model to reduce time consumption.

Contents

1 Introduction

Donald Hebb's 1949 book, The Organization of Behavior, proposed a connection between the firing activities of neurons in the brain and cognitive function. Over the years, researchers have discovered many facts about this relationship. In 1971, O'Keefe and Dostrovsky discovered place cells in the hippocampus of rats, which fire when the animal is in a specific location in its environment[1]. Other types of cells, including head direction cells[2], grid cells[3], and others[4][5], were subsequently discovered, each reflecting different types of spatial information in the environment.

These discoveries have led to fruitful collaborations between computer science and neuroscience, such as the Tolman-Eichenbaum-Machine (TEM)[6]. TEM uses Hebbian learning to combine relational memory with spatial cells, and can even make inferences on non-spatial tasks. By mapping non-spatial tasks to the spatial structure represented by spatial cells, TEM can generalize structural cognition across different yet similar contexts[7]. The potential of such biologically inspired methods has been demonstrated in numerous studies.

On the other hand, Moravec's paradox highlights the difficulty of conventional methods in giving computers the same skills as a one-year-old, despite their adult-level performance in other tasks[8].

Efforts have already been made to construct navigation systems using biologically inspired methods.In 2015, Edvarsen constructed an offset detector network, a pre-wired neural network that calculates the relative direction between two positions represented by firing of grid cells[9]. In 2019, Edvardsen proposed a navigation system, which relies on hippocampal replay on place cell maps to avoid obstacles[10]. During hippocampal replay, the system traces back from the goal location to the current location according to the topological information in the place cell map.

Erdem and Hasselmo suggested doing "look-ahead probes" in the grid cell system, until useful information can be retrieved from the place cells that are activated by these probes[11]. This method is called linear look ahead(LLA). Tim Engelmann constructed a navigation method based on LLA, which navigates through a maze by performing LLA on a pre-constructed cognitive map. Although it has potential, this method has drawbacks such as limited flexibility in dynamic environments, low accuracy in navigation, and high time consumption.

In this paper, we propose a method called BioNav, which addresses the limitations of the baseline method constructed by Tim Engelmann[12], is adaptive to dynamic mazes, and consumes less time during navigation. We will compare BioNav to a D* Lite to demonstrate the strengths and weaknesses of biologically inspired methods versus conventional methods. This paper begins by introducing some basic concepts in biologically inspired methods, followed by a brief overview of the baseline method and its weaknesses. We then introduce

our improvements and conduct comparison experiments between BioNav and conventional methods.

2 Background and Concepts

In this chapter, we will describe the relevant cell types and concepts used in our work. These concepts include cell types used in this papers, basic concepts in biologically inspired methods, and concepts from conventional navigation methods that will be used for comparison.

2.1 Grid Cells

Moser and Moser discovered cells in the medial entorhinal cortex (MEC) of the rat brain that would fire at several equally-spaced locations in the environment [3]. Those cells, namely grid cells, form a universally applicable spatial coordinate system[13]. As a rodent traverses a linear path, as shown in Figure 2.1, an individual grid cell will fire at equally spaced locations. These locations are a specific distance apart, the grid-scale, and are independent of the speed or path that the rodent chooses. To understand how grid cells form a coordinate system, let us first consider a single grid cell. We denote its grid-scale with λ and propose the grid cell fires at the position x_0 where the rodent begins its line traversal. Whenever the cell fires, we know the rodent must be roughly at a multiple of the grid-scale, so $x = x_0 + k \cdot \lambda$, where $k \in N$ is property splits the traversed line into units of grid-scale size. If the grid cell fires, we know the rodent is positioned at the beginning of a line-unit. Next, we combine several grid cells with the same scale but shift their firing peaks by an offset of ϕ. If we choose ϕ small enough, there will always be a grid cell firing. When the n-th grid cell fires, the rodents position can be described as $x = x_0 + n \cdot \phi + k \cdot \lambda$. Therefore, based on which grid cell fires we can determine the rodents relative position within a line unit.

2.2 Place Cells

O'Keefe and Dostrovsky were the first to discover a correlation between cell firing in the rat's hippocampus and the current location of the rat[1]. O'Keefe investigated the properties of those cells further and named them "place cells". Place cell firing is especially strong in a particular area of the environment called a place field. In contrast to grid cells, that fire at several locations in the environment, a place cell only fires within a singular place field. However, they can be reused across different environments. Place cells form the basis of mapping the environment to a cognitive map, as first introduced by Tolman[14].

The primary input to the place cells are believed to be grid cells. Many researchers suggested that the combined signals of grid cell modules with different scales represents the signal of place cells[15]. However, further researches on young rats, who have not developed

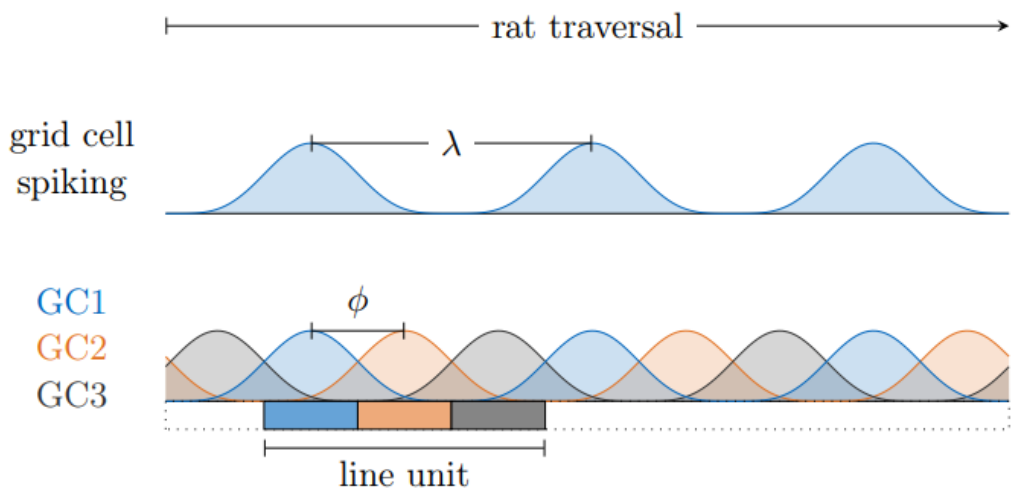


Figure 2.1: This figure shows how do individual grid cells work together to form a grid cell module. As the rat traverses a linear path, a grid cell will spike every time after traveling a distance of λ. By combining several grid cells (GC) with the same grid-scale λ but a phase offset of ϕ, the space is made discrete.[12]

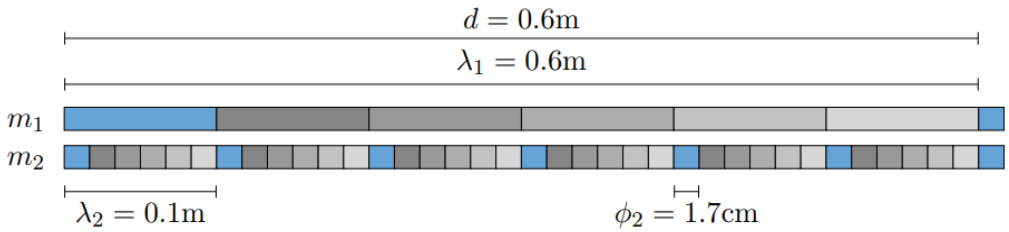
grid cell system yet, indicate that other cells like border cells are also relevant in the formation of signals of place cells[16].

2.3 Prefrontal Cortex Cells

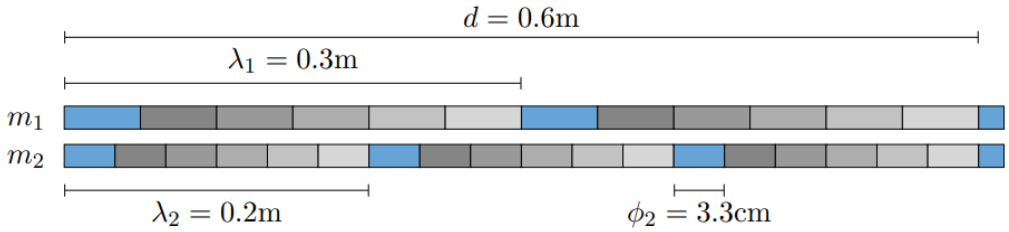
The prefrontal cortex is closely connected to the hippocampus, and together they are believed to be capable of "memory encoding and context-dependent memory retrieval" [17]. Storing information about the environment in the form of a "cognitive map" is necessary to perform topology-based navigation. Erdem and Hasselmo proposed a navigational model with a variety of cell types located in the prefrontal cortex, namely recency cells, topology cells, and reward cells[18]. While the existence of these particular cell types lacks biological evidence, there are recordings implying that reward information could very well be stored in the medial prefrontal cortex and retrieved for goal-driven navigational tasks[19][20][21].

2.4 Two Flavours of Navigation

There are two flavours of navigation: vector based navigation and topology based navigation. They both have their advantages and disadvantages. Figure 2.3 illustrates the task and ideas behind these two methods of navigation.

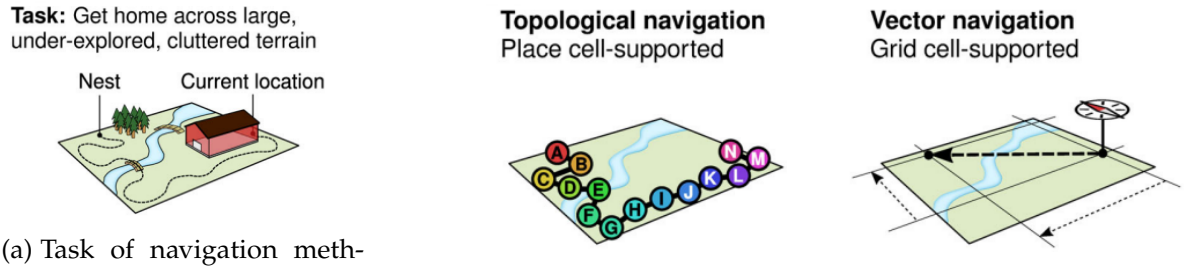


(a) Nested approach.



(b) Combinatorial approach

Figure 2.2: Discretization of the one dimensional space by a combination of two grid cell modules, m_1 and m_2. The colors indicate which of the six grid cells within each module is most active at the respective location. In this example, both approaches discretize the axis for a distance of d unambiguously. (a) m_1 has a relatively large grid-scale and defines the environment size. m_2 discretizes the space further and increases the resolution. Note that usually grid-scales would differ only by factor of about 1.4, but this combination highlights the difference to the combinatorial approach better. (b) The combination of the two grid cell modules forms a grid code. The code is unique across a distance defined by the common multiple of the grid-scales and the number of grid cells per module. Adding another module would greatly increase the environment size that can be converged.[12]



(a) Task of navigation methods.

(b) Two flavours of navigation.

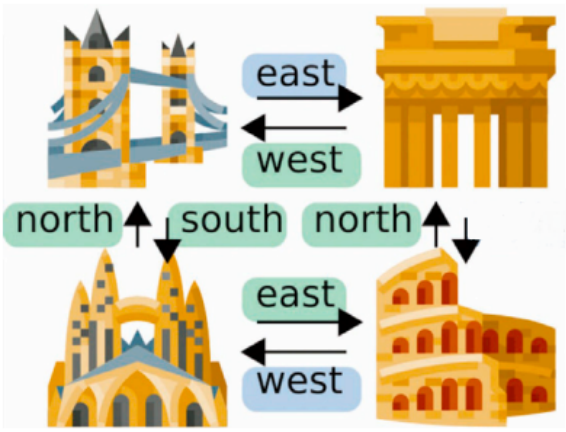
Figure 2.3: This figure shows the two flavours of navigation: topology based navigation. Using topological navigation the agent builds up a graph of nodes representing places from experience or knowledge, the nodes are connected using rules, for example the order visit during an exploration, or the distance on a map. The agent access the constructed graph and acquires the guidance for the next step from the topological information. During vector based navigation, the agent only uses the sensory of the current location and the sensory or knowledge of the goal location. The agent calculates the relative direction between its current direction and the goal location. The agent acquires an overview and the shortest path to the goal location.[7]

2.4.1 Vector Based Navigation

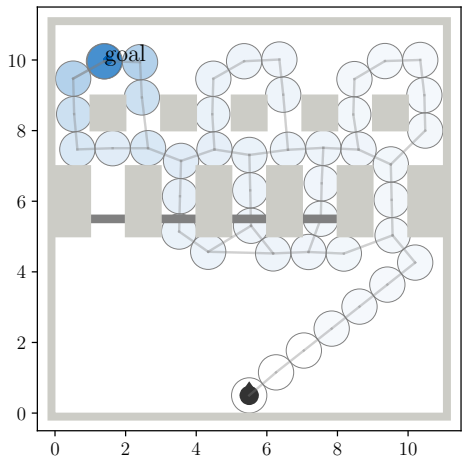
Vector based navigation uses less information than topology based navigation. Vector based navigation usually only need the sensory data of the current position and the knowledge about the goal position as input. The output is the relative direction between the current position and the goal position. This method of navigation provides the shortest path to the goal and the most general guidance to the solution of the navigation problem. However, this method disregards the obstacles between the two obstacles, for example a river. In figure 2.3, vector based navigation will provide the agent a path across a river to the goal. This path is the shortest but is infeasible because it disregards obstacles.

2.4.2 Topology Based Navigation

Topology based navigation need much more information than vector based navigation. Topology based navigation not only needs the information about the current position and the goal location, but also the knowledge about experienced locations and relations between them, from which the agent calculate possible paths to the goal. This method usually does not return the most optimal route because it is constrained by the previous knowledge and ignores possible shortcuts. However, this method provide paths, were feasible from the current knowledge. In figure 2.3, topology based navigation provide the agent a feasible path over the bridge across the river, which the agent has experienced from before. However, this path is not optimal, because from node A to node D there is a shortcut, but because the agent has no experience of this shortcut, there is no direct connection between A and D, thus



(a) Cognitive map containing landmarks and its relative allocentric positions[6].



(b) An example of cognitive maps used in this paper.

Figure 2.4: This figure shows the idea of cognitive map and an example of cognitive map used in this paper. Figure 2.4a is a general cognitive map, where different places and their correlations are recorded. The agent should be able to utilize this kind of cognitive map and solve navigation problems. Figure 2.4b show an example of cognitive maps used in this paper. The gray areas are the obstacles. The circles are markers for different places, which are represented by different place cells in this paper. The edges connecting the circles defines the topological relation between them, which is generated through calculating the order of visit during exploration. Different shades of blue marks the reward signals of each place, which is correlated to the topological distance to the goal. The circle with the deepest blue marks the goal in the maze.

topology based navigation can not utilize this shortcut.

2.5 Cognitive Map

A cognitive map represents your knowledge of an environment. It is built up from the experiences in an environment. It can contain information of the environment in many aspect. For example, it can contain landmarks, position relations between places, temporal information about the latest visit, and even the experiences of single places[7]. The cognitive map takes different forms in different method setups. In the setup of this paper, the cognitive map consists of connection setup between place cells and grid cell system, initialization of grid cell system and prefrontal cortex cells. This will be discussed in details in chapter 3.

Linear Look Ahead is method used during navigation to utilize information in the cognitive map. The main idea is to imagine going to a place and its benefits before actually going there. The results gained from these imagining behavior provide important guidance when making

decisions. For example, if one hungry person imagines going to a supermarket at mid-night, he will reach the conclusion that it will not be beneficial for him as he knows from experience that the supermarket will be closed. This will lead to his decision to go to other places which have a better imaginary results from his knowledge, for example a night bar.

2.6 Conventional Methods

Before the attempts of using the discoveries in neuroscience to create methods that navigate in environments, there has already been a lot of well established methods that solves the problems of navigation in dynamic environment. Among these methods, there are two types of methods that will be mentioned in this paper. They are sampling based method and grid based method. These methods will be explained and compared in chapter 8. From these methods, the method of D* Lite[22] will be selected to represent the general conventional methods and compared with the final version of our method, BioNav through experiments in this dynamic maze.

There are also other types of navigation method, for example optimization based methods and visual trajectory based methods. However, these methods either does not suite our experiment setup or the method has very little similarities with BioNav, which could not produce meaningful comparison results.

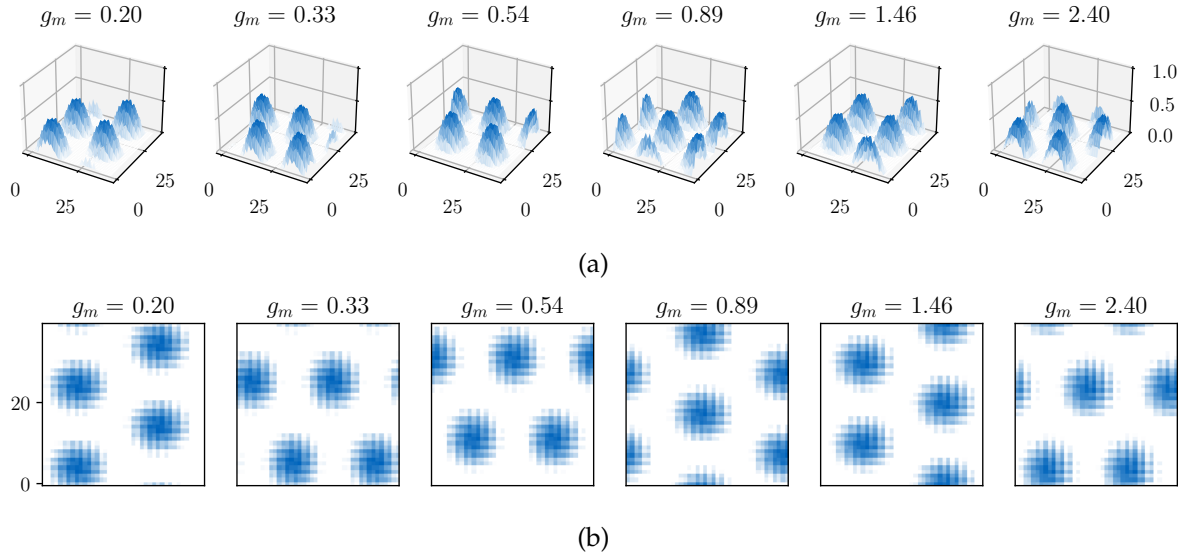


Figure 3.1: Figure a) shows the 3D plotting of the neural sheets in each Grid Cell module with the respective gm value after random initialization. As shown in the 3D plotting, the peaks are organized in a periodic manner while having the highest firing value in the center of each peak and then gradually decrease to zero in a slope. Figure b) shows the 2D plotting of the neural sheet, which can be interpreted as looking at the 3D peaks from above. As is shown in the 2D plotting, the firing pattern of the neural sheet of the Grid Cells form a hexagonal pattern, which matches the firing patterns of grid cells in experiments mentioned in the introduction chapter.

3 Demonstration of the Baseline Method

3.1 Overview of the Baseline Method

3.1.1 Grid Cell System

In the baseline method, the formation of the neural sheets of Grid Cells are well implemented. As you can see in figure 3.1, the neural sheets of each Grid Cell module show clear hexagonal firing patterns after random initialization(initialize using random direction vectors). The grid cell modules in the baseline method follows the implementation proposed by Edvardsen[9] and related works[23].

3.1.2 Place Cell System

When a place cell is formed, it takes a snap-shoot of the firing patterns of the neural sheets of the grid cell system. The current firing value of each grid cell now becomes the connection weight of the created place cell to this grid cell. This snap-shoot represents an allocentric position in the environment. The firing value of the place cell is calculated by adding up the multiplication of the connection weight of each grid cell to this place cell and the current firing value of each grid cell. In another word, place cell returns a higher firing value whenever its snap-shoot and the current firing patterns of the grid cell system become more similar, and returns a lower firing value when there are bigger differences between them.

3.1.3 Cognitive Map

A cognitive map of an environment is an organization of place cells and consists of two types of information: neighborhood information and reward information. Neighborhood information contains which pairs of place cells are neighbors to each other. Reward information defines the goal place cell as having the highest reward, while the rewards of other place cells are negatively correlated to the topology distance to the goal place cell in the neighborhood information.

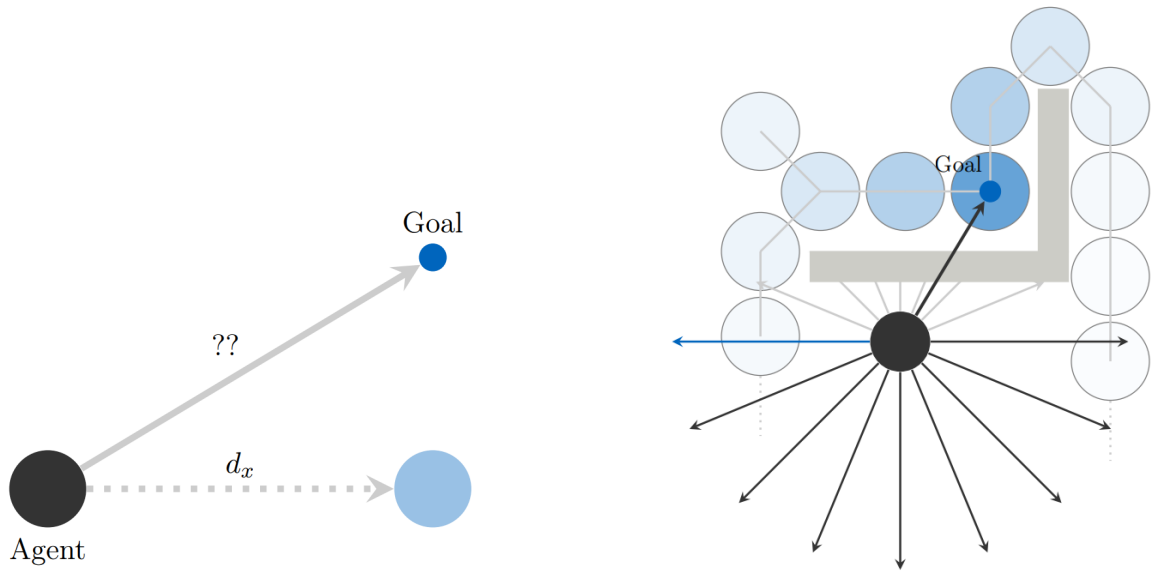
3.1.4 Linear Look Ahead

Linear Look Ahead(LLA) is the key method to utilize the information provided in the cognitive map. During Linear Look Ahead, the agent "imagines" going in a specific direction and "imagines" the rewards of going in that direction. Then the agent decides on the next move according to its "imaginations". To be specific, the agent first stores the current state of the grid cell system. Then, the agent performs virtual updates in one direction to its grid cell system(updating the system without taking an actual move in that direction). The agent checks the firing values of its place cells on these virtual locations, and then remembers the rewards of these virtually firing place cells. After the virtual update, the agent restores the states of the grid cell system.

There are two types of LLA, corresponding to the two flavors of navigation.

Projective Linear Look Ahead(Projective LLA), as is shown in figure 3.2a, corresponds to the vector based navigation. The agent first selects a goal place cell. It then performs a Linear Look Ahead in each of the axes in the 2D plane. The agent compares the firing patterns in the virtual neural sheet with the snap-shoot of the selected place cell in a projective manner, and then decides on distance to travel on each of the axes based on the firing value in the Linear Look Ahead. This would return a vector in the 2D plane, which represents the allocentric vector from the current position of the agent to the location represented by the selected place cell.

Directed Linear Look Ahead, as is shown in figure 3.2b, corresponds to the topology based navigation. The agent does not have a pre-selected place cell. It performs a Linear Look Ahead to all allocentric directions that are not blocked by observable obstacles. It then selects



(a) Projective LLA.

(b) Directed LLA.

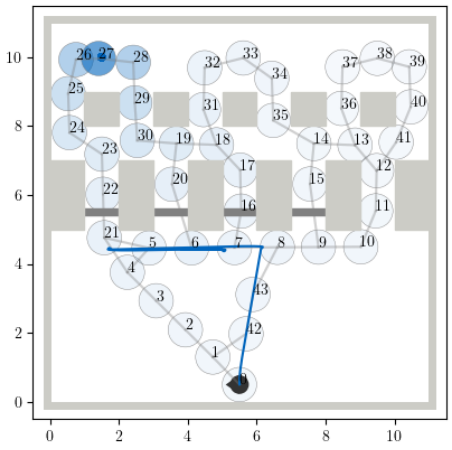
Figure 3.2: This figure shows the two kinds of linear look ahead(LLA). Figure a) shows the Projective Linear Look Ahead(PLLA), during which the agent performs a virtual update along each of the axes in the plane. The outcome of the PLLA is the relative direction vector from its current location to the goal location. In figure a), the agent performs virtual updates along the X axis and calculates the relative projective distance between its current location and the location of the goal on the X axis. Figure b) shows the Directed Linear Look Ahead, during which the agent performs virtual update along multiple unblocked directions. In figure b) the agent does not perform virtual updates in directions that are blocked by the obstacle(marked with gray), even when one of the blocked directions includes direction to the goal. The agent selects one direction from the remaining directions that has the best reward. The selected direction is marked by blue.[12]

the direction with the highest reward. This corresponds to the topology based navigation because it is attracted to the higher reward signals, which is defined by topology distances to the goal place cell.

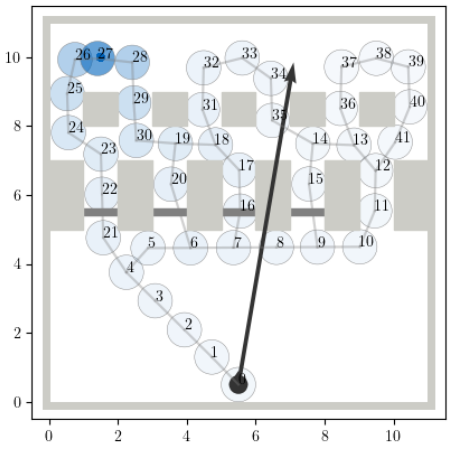
3.2 Performance of the Baseline Method

3.2.1 Navigation Conducted by the Baseline Method

The baseline method combines both vector based navigation and topology based navigation. The baseline method first constructs a cognitive map of the maze during exploration. During navigation, the agent uses both navigation methods to utilize information from the constructed



(a) The baseline method can not deal with changes in door setup.



(b) The baseline method is inaccurate in projective linear look ahead.

Figure 3.3: This Figure shows a series of experiments demonstrating the weaknesses of the baseline method. In figure 3.3a, the agent has a cognitive map generated by going through all the pathways. However, the agent is not able to select the correct pathway to the goal when the setup of the doors has changed. In figure 3.3b, the baseline method can not correctly identify the vector to the goal during the vector based navigation. This shows that the implementation of vector based navigation in the baseline method is inaccurate.

cognitive map. At the start of the navigation, the agent performs a Projective LLA to determine the distance and direction of the goal to its current position. It then goes straight in the direction of the goal. If it encounters an obstacle blocking the way, it switches to Directed LLA to determine a subgoal. It continues with Directed LLA until the agent senses that the goal is near. It performs a Projective LLA in the end to reach the goal.

3.2.2 Shortcomings

Although the baseline is able to navigate in dynamic environments, it suffers from the following shortcomings...

The baseline method is unable to deal with changes in the environment. The baseline method is not able to deal with changes in the environment. If the doors in the maze are opened during the exploration phase but closed during the navigation phase, the agent is unable to find the path to the goal, even though it has explored other paths in the environment.

The vector based navigation of the baseline method is inaccurate. The projective LLA in the baseline method is poorly proposed and is error prone. As is shown in figure 3.3, the agent can not correctly find the vector to the goal place cell during vector based navigation. This will be discussed further in the chapter 5.

The time consumption in the baseline method is too huge. The speed of the baseline method is very slow. A navigation through a standard maze takes half an hour. This potentially makes the method not competitive to conventional methods like D* Lite.

It lacks comparison experiments between the baseline method and the conventional methods. To show the potential of BioNav, it needs to be compared to the conventional methods and prove through experiments that BioNav has functionalities that are not provided in the conventional methods, while the shortcomings of BioNav can either be dealt with or are not lethal.

4 Flexibility in Dynamic Maze

As is shown in chapter 3.2.2, The baseline method is not able to deal with changes in the environment very well. In Figure 3.3a, the agent was not able to identify that the doors are closed and choose to go to another sub-goal that it has not visited. This is because when the agent performs a linear look ahead, the reward signal of the place cells at gate 2 is the strongest. Every time the agent performs a linear look ahead, it is attracted to the place cells at gate 2.

4.1 Cause of the Problem

After careful examination, the cause of these problems can be formulated as follows. The baseline method does not recognize the visited doors are closed during navigation. The baseline method also does not utilize the information in the recency cell, which records the most recent visit of a place cell by the robot, during topology based navigation. This causes the agent to keep returning to a place that it has visited before.

4.2 Proposed Solutions

In order to address these problems, we have proposed several potential solutions. Among them are the creation of block cells and a visited cell list. Between these two solutions, the block cell wished to address the problem that the agent does not recognize the obstacles(a closed door in this case) during navigation. However, the solution of block cells is later abandoned, because it creates too much computation burden during navigation. On the other hand, the second solution, which wished to address the problem of the agent returning to visited places, has proved to be effective. The second solution has even surprisingly proved to be effective even in addressing the problem of dealing with closed doors, which the block cells wished to address. Thus, in the end, only the implementation of the visited list was adopted.

4.2.1 Block Cells

To address the problem of not being able to identify closed doors, we proposed to add a new class of cells that represent obstacles in the environment. This type of cell corresponds to the landmark cell in the brain. A block cell has the same functionality of a place cell, except that it has a shorter firing range than a place cell(firing threshold is set as 0.95 instead of 0.85 for place cells). A block cell also stores a snapshot of the grid cell modules. The difference is that

it updates its grid cell modules virtually before the creation of the block cell to the direction of the obstacle, so that the block cell is placed upon the allocentric positions of the observed obstacles. To describe an obstacle, a very large number of block cells is needed, because a block cell has a very small firing range. If the firing range increases, the agent will receive block cell signals at the center of the path, which is not desirable.

However, the main reason for me to abandon this method is that maintaining so many block cells will put a huge calculation burden on the agent. For every update of the agent, all the place cells and block cells need to be compared with the current grid cell modules and compute their firing values. After adding block cells, the time of a single run is increased from half an hour to six hours. This makes further explorations of this method difficult and the adoption of block cells in the BioNav impractical.

4.2.2 The Visited Cells List

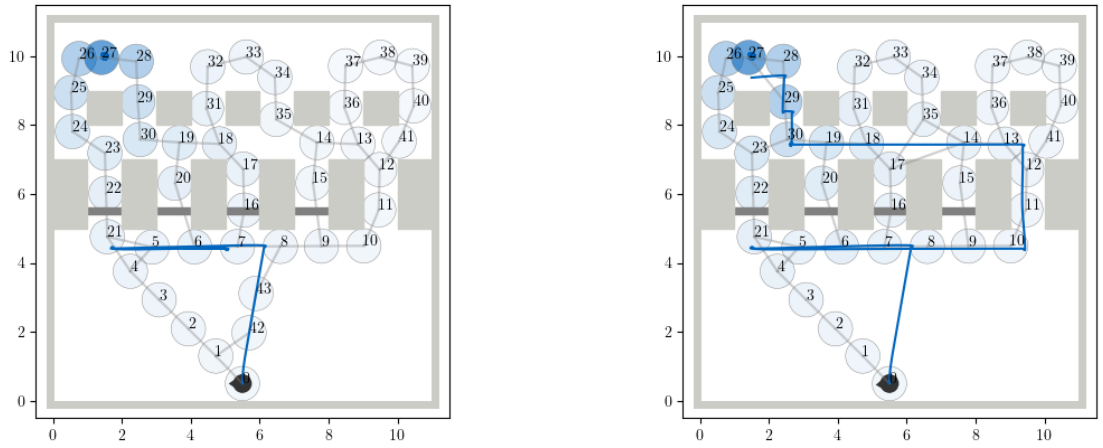
The second proposal that we made to solve the problems of navigating in a dynamic environment is to keep a list of visited place cells. In another word, if a place cell fires above the firing threshold, this place cell is added into a visited list and its reward signal is excluded during the topology based navigation, which is linear look ahead in this case.

This method also has biological plausibility. The method creates a feedback channel from recency cells to reward cells during topology based navigation. If the recency cell is invoked, it will have a constant inhibition signal to its reward cell, which excludes the corresponding place cell from the linear look ahead process. After excluding a place cell from the linear look ahead, the agent will go to other place cells because there are higher reward signals during linear look ahead.

This method proves to be not only useful in avoiding the agent from being attracted to the visited place cells, but also useful in avoiding obstacles. Because if a place cell is visited when the location represented by the place cell is explored during navigation. If the visited place cells are excluded during the path finding process, the agent will explore the locations represented by other place cells instead. If all the place cells are visited and the agent still has not reached the goal, it means it is impossible to reach the goal using the current cognitive map. Thus, visited cells list also solves the problem of obstacle avoidance by encouraging the agent to explore all the possible routes recorded in the cognitive map during the exploration.

4.3 Solution Results

After adding the visited cell list to the method, the problem that agents are trapped in one visited place does not happen anymore in this environment setup. The difference after adding the visited cell list is shown in figure 4.1. The agent is able to avoid obstacles and adapt to changes of the doors setup. This is because the cells that exceeded the firing threshold once are excluded from Linear Look Ahead, which is done by setting the reward signal to zero. In this way, the agent is discouraged from visiting the place cells that have already been triggered.



(a) The baseline method without the visited cell list.

(b) The baseline method after adding the visited cell list.

Figure 4.1: This figure shows the difference between The baseline method without visited cell list and with visited cell list. The agent uses the same trajectory during exploration and then tries to navigate to the goal with a changed setup of doors. Figure 4.1a shows the navigation trajectory before adding visited cell list. The trajectory shows that, The baseline method can not adapt to the changed setup of doors and is not able to reach the goal. The baseline method keeps returning to the visited places. Figure 4.1b shows the navigation trajectory after adding visited cell list. The agent explores possible doors and finds the goal during navigation.

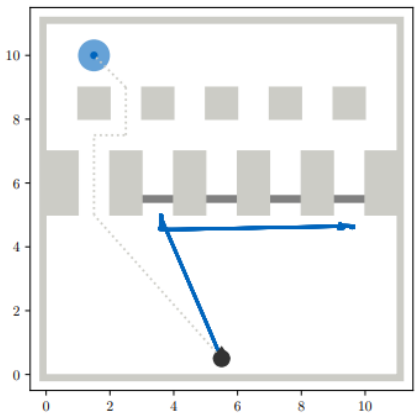


Figure 4.2: This figure shows that the agent can not go through door 1 to get to the goal, because door 1 is not explored in the exploration phase and thus unknown to the agent. This problem remains unresolved even after adding a visited cell list.[12]

However, the agent still does not have a sense of obstacles in its cognitive map. The obstacle avoidance is done by encouraging the agent to explore other areas and other routes that it has recorded in the map, when the previously planned route turns out to be blocked.

Moreover, as is shown in figure 4.2, even after adding the visited cell list, the agent is still not able to go through the areas that are not explored during the exploration phase.

As is shown in figure 4.1, the problem of identifying the false direction during the vector based navigation phase remains unresolved by visited cell list, because the agents are going in the wrong direction at the start of the navigation. This problem will be analyzed and dealt with in the chapter 5.

5 Projective Linear Look Ahead

5.1 Functions of Projective Linear Look Ahead in BioNav

To give the agent the ability to navigate in the environment using the prior knowledge from a cognitive map, only using the Directed Linear Look Ahead(Directed LLA), which represents topology based navigation, is not good enough. Even though there are methods of navigation which uses only topology based navigation, like Dijkstra(algorithms that uses a heuristic function, for example A*, implies the use of vector based navigation). The agent needs to have the ability to “imagine” the direction to a chosen place, so that the agent could have a general idea about where the goal is, which will help to boost the performance of the whole algorithm.

The Vector based navigation in BioNav has two functions:

- The agent goes in the direction of the goal directly in the beginning of the navigation.
- The vector based navigation solves the "last mile" problem in the navigation. Which means if the agent senses that the goal is close enough, it turns to the vector based navigation.

In conventional navigation methods, the vector based navigation is embedded into the topology based navigation, and provides hints and set up maps, which are used by the topology based navigation. A* uses the vector distance to the goal as a heuristic function to optimize the order of expanded nodes. Probabilistic Road Map uses the distance between each pair of sampling nodes and judges whether the connection of two nodes is blocked by obstacles to construct the edges in its roadmap. This is discussed in more detail in chapter 8. It is to be admitted that the potential of the vector based navigation is not very well explored in current BioNav. However, it is still providing distinct functionalities to BioNav, which is shown in chapter 9.

5.2 Implementation of Projective Linear Look Ahead in BioNav

Algorithms 1 shows the workflow of Projective Linear Look Ahead. The agent performs virtual updates along two axes of the plane(in four directions). On each step of virtual update compute the projective firing of current grid cell system firing to the targeted place cell. If the firing of one step exceeds the threshold of 0.95, this step is recorded as the look ahead result in that direction. If there is no step that exceeds the threshold, then the step with the maximum firing value is recorded.

The method to compute projective firing values is demonstrated in figure 5.1a. The projective firing patterns are produced from the firing patterns of the agent's grid cell modules by summing over rolls or columns. The projective firing value is produced by computing the similarity values between the projective firing patterns of the current grid cell modules and the projective firing patterns of the snapshot stored in the place cell that represents the goal.

$$firing = \frac{p_g \cdot p_p}{p_p \cdot p_p} \tag{5.1}$$

In this equation, p_g is the projective firing pattern of a grid cell module, and p_p is the projective firing pattern of the snapshot of this grid cell module stored in the place cell representing the goal. The firing is around 1.0(inaccuracy could appear during float number computations), when the virtual position of the agent has the same projective position with the place cell.

5.3 Analysis of the Problem

As is shown in algorithms 1, the accuracy of the vector based navigation heavily depends on calculation of projective firing between the goal place cell and the virtual grid cell system. The figure 9.14d shows the calculation of projective firing. The two projective patterns are only distinct enough in one projection direction, which shows peaks in the firing pattern. However, if we look at the projective firing patterns of the grid cell modules in The baseline method, only four modules show peaks in the projective direction of the x axis. If we sample the projective firing between the goal place cell and grid cell system from different locations in the maze. It is clear that the projective firing values are more relative to the projective distances on y axis than on x axis. This corresponds to the problem that the vector based navigation is inaccurate on the x axis.

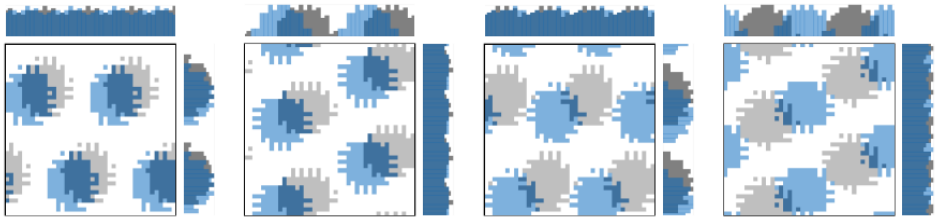
5.4 Fixing the Problem

To fix the problem, we need to have a grid cell system that shows sufficient accuracy on both x direction and y direction. To achieve this, the grid cell modules in the grid cell system need to have enough modules, whose projective firing patterns show clear spikes on x axis and also have enough modules, whose projective firing patterns show clear spikes on x axis.

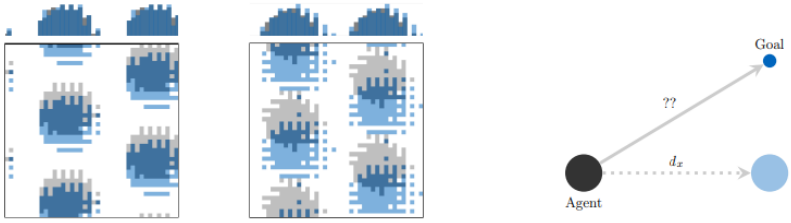
To initiate such a grid cell system, we need to know what factors are affecting the spike formations of the projective firing patterns of grid cell modules.

5.4.1 Factors Affecting The Formation of Spikes of Projective Firing Patterns of Grid Cell Modules

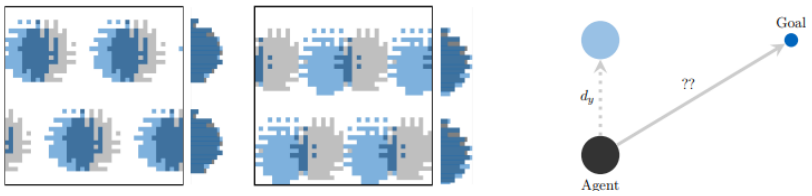
After experiment, we found that both the g_m value of the grid cell module, which denotes the update scale, and the arrangement of the heading directions affects on which axis clear spikes appear in the projective firing pattern of grid cell modules.



(a) Projective firing pattern of place cell and grid cell modules.



(b) Projective Linear Look Ahead on the X axis.



(c) Projective Linear Look Ahead on the Y axis.

Figure 5.1: This figure shows how projective firing is computed. Figure 5.1a shows the projective firing pattern of grid cell modules. As demonstrated in the figure, the pattern is produced by summing the firing values over rows or columns in the neural sheet. The gray patterns are the snapshots stored in a place cell. The blue patterns are the firing patterns of grid cell modules. The projective firing patterns show clear spikes in some directions while showing no clear spikes in other directions. Figure 5.1b shows the projective linear look ahead on the x axis. The agent performs virtual updates on the x axis, while comparing the projective firing pattern of the virtual grid cell modules with the projective firing pattern with the goal place cell. The projective firing value is the highest when the spikes in projective firing patterns coincide with each other, as demonstrated in the figure. Figure 5.1c shows the projective linear look ahead on y axis[12].

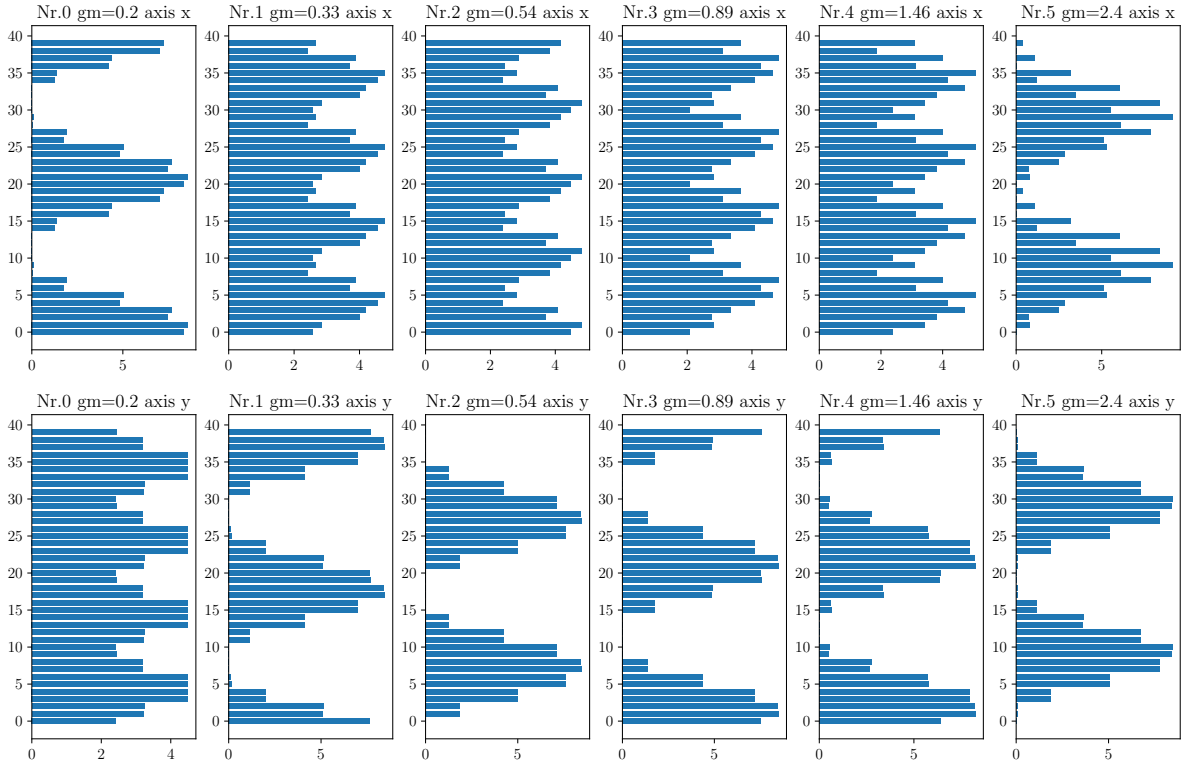


Figure 5.2: This figure shows the projective firing pattern of each grid cell module in The baseline method. The upper roll is the projective firing pattern of the x axis, while the bottom roll shows the projective firing pattern of the y axis. The projective firing pattern only shows clear spikes in two of the grid cell modules on the x axis, while showing clear spikes on five of the grid cell modules on the y axis. This explains the problem illustrated in figure 3.3b, where the agent can calculate the vector based navigation with accuracy only on y axis, while showing huge mistakes on x axis.

Algorithm 1 Pseudo code for Projective Linear Look Ahead

Require: Grid Cell System, Place Cell(Goal), Cognitive Map
  $t \leftarrow$ update time
  $d \leftarrow (0,0)$ ▷ stores the result of Projective Linear Look Ahead
  $f \leftarrow (0,0)$ ▷ stores the result firing of Linear Look Ahead
  Reset virtual Grid Cell System firing
  Define the two look ahead axis vector $\{\vec{e_1}, \vec{e_2}\}$
  Define the four look ahead speed vectors $V = \{k\vec{e_1}, k\vec{e_2}, -k\vec{e_1}, -k\vec{e_2}\}$
  for $\vec{v} \in V$ **do**
    for in numbers of look ahead steps **do**
      $p_1 \leftarrow$ compute_projective_firing(GC system, Place Cell)
      if $p_1 \geq$ Look Ahead threshold **then**
        record $k\vec{e_1} \cdot t$ in respective axis in d
        record p in respective axis in f
        Break
      else if $p \geq f$ **then**
        record $k\vec{e_1} \cdot t$ in respective axis in d
        record p in respective axis in f
      end if
      virtually update Grid Cell System by $\vec{v} \cdot t$
    end for
  end for
    return d

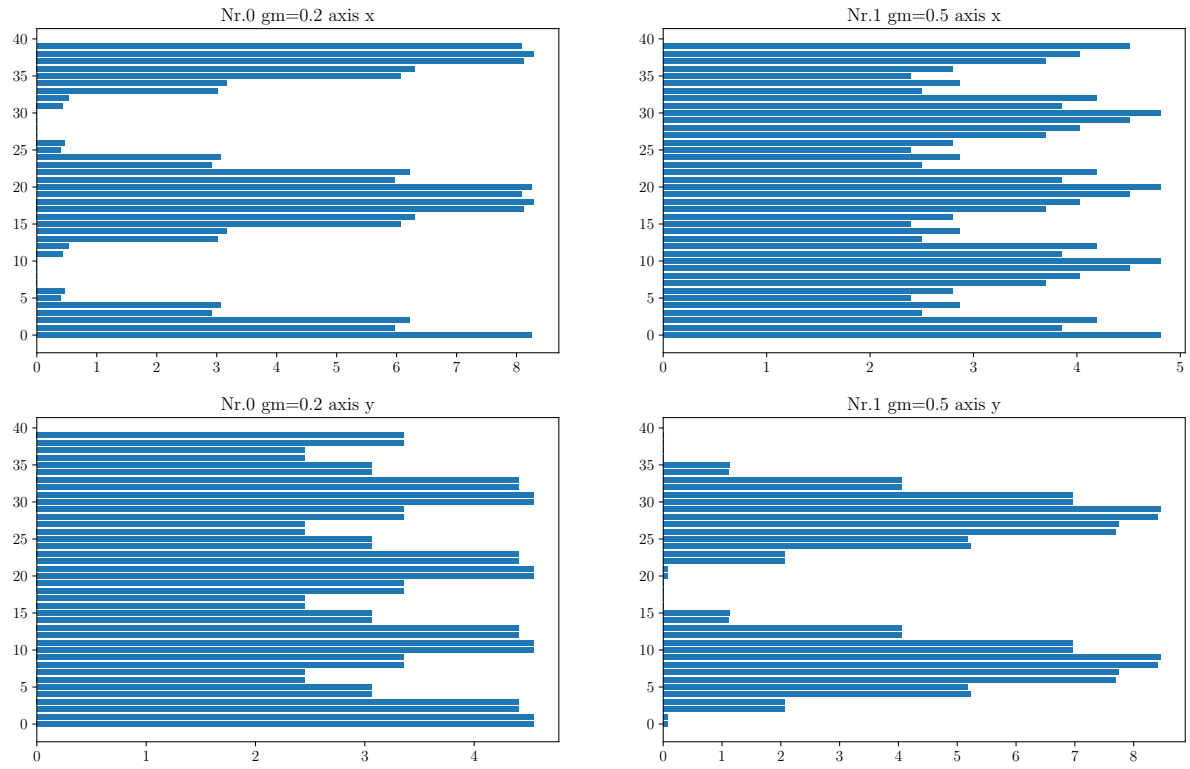


Figure 5.3: This figure shows that the grid cell module shows clear spikes on different axes when the g_m value changes. On the left is the projective firing pattern of a grid cell module with g_m value of 0.2, and on the right is the projective firing pattern of a grid cell module with g_m value of 0.5. This figure shows that the grid cell module with g_m value 0.2 shows clear spikes on the X axis while the grid cell module with g_m value of 0.5 shows clear spikes on the Y axis.

Figure 5.3 shows that the grid cell module using the same setup but with different g_m values show clear spikes on different axes in their projective firing patterns.

Figure 5.4 shows that grid cell modules using different alignment of heading directions show clear spikes on different axes in their projective firing patterns. The heading direction of a grid cell is a direction vector $\vec{e_\theta}$, the grid cell is more sensitive when the angle between the speed signal and the heading direction becomes smaller.

5.4.2 Rearrangement of Grid Cell Module to Create Diversity

Figure 5.5 shows the result of the projective firing patterns in the grid cell system after we arranged the g_m values and alignment of heading direction. In our arrangement, we used seven grid cell modules to compose a grid cell system(one module more than the baseline method). The g_m value is computed using the following equation proposed in Edvardsen's paper[9].

$$\begin{aligned} R &= \sqrt[M-1]{g_{max}/g_{min}} \\ g_m &= g_{min} \cdot R^{m-1} \end{aligned} \tag{5.2}$$

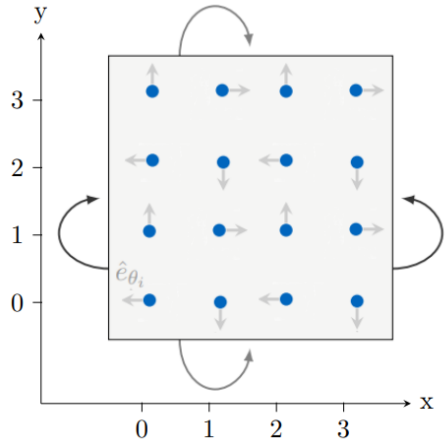
In this equation g_{min} and g_{max} represents the biggest scaling factor and the smallest scaling factor of grid cell modules in the grid cell system. M represents the number of grid cell modules, while m represents the index of the current grid cell module in the system starting from 1.

In our arrangement, the $M = 7$, $g_{min} = 0.2$ and $g_{max} = 2.4$. The heading alignment is an alternation with figure 5.4a and figure 5.4b starting from figure 5.4a.

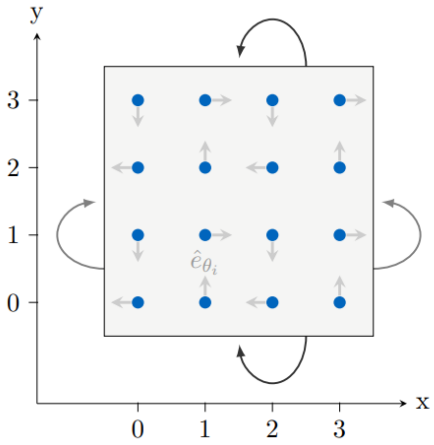
Using our arrangement of the grid cell system, the accuracy of the projective firing is much better than using the implementation from the baseline method. This is demonstrated in figure 5.6. In the comparison, the positions where the projective firing value over 0.90 stays within $0.5m$ range in BioNav, and the projective firing value over 0.90 is around the range of $4.0m$ in the baseline method.

5.4.3 A New Method for Calculation of Weight According to the Scales of Each Grid Cell Module

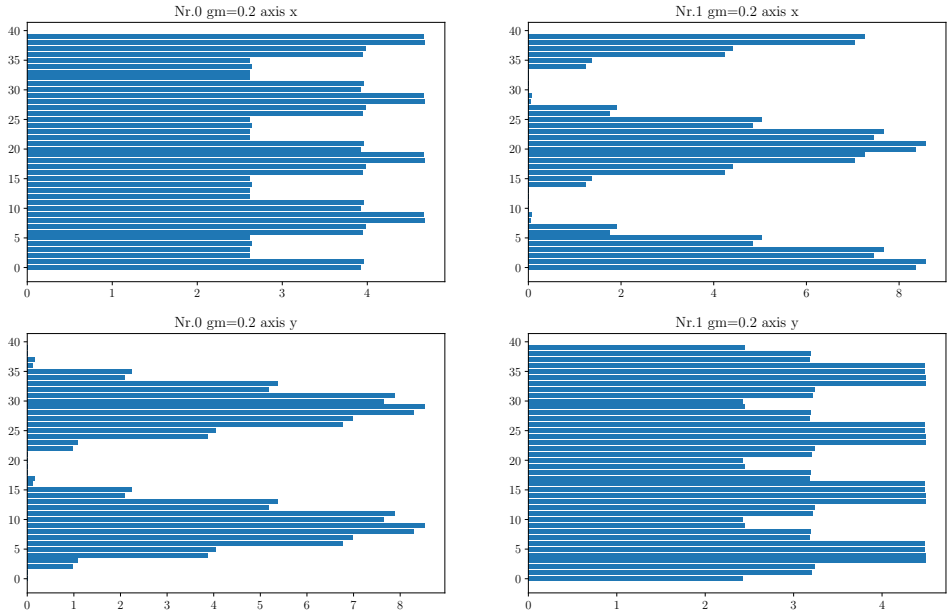
Moreover, in the baseline method, the weight of grid cell modules with different scales are the same, when computing the projective firing value. This is not suitable because the grid cell module with a larger scale(smaller g_m value) represents a larger area with less precision. Thus, the difference of grid cell modules with larger scale should have more weight than the grid cell modules with smaller scale. This can be more intuitively understood as higher digits represent more value in a number than lower digits. Following this logic, we proposed the following equation for the calculation of the weight:



(a) Heading alignment Nr.1



(b) Heading alignment Nr.2



(c) The projective firing patterns of the grid cell system.

Figure 5.4: This figure shows that the heading alignment of the grid cell in a grid cell module also decides on which axis the projective firing pattern shows clear spikes. Figure 5.4c shows two grid cell modules using the same g_m value($g_m = 0.2$) but using different heading direction alignments show clear spikes on different axes. Left hand side of the figure 5.4c shows the projective firing pattern of a grid cell module using the heading direction in figure5.4a. Its grid cell module shows clear spikes in its projective firing pattern on the x axis. Right hand side of the figure 5.4c shows the projective firing pattern of a grid cell module using the heading direction in figure5.4b. Its grid cell module shows clear spikes in its projective firing pattern on the y axis.

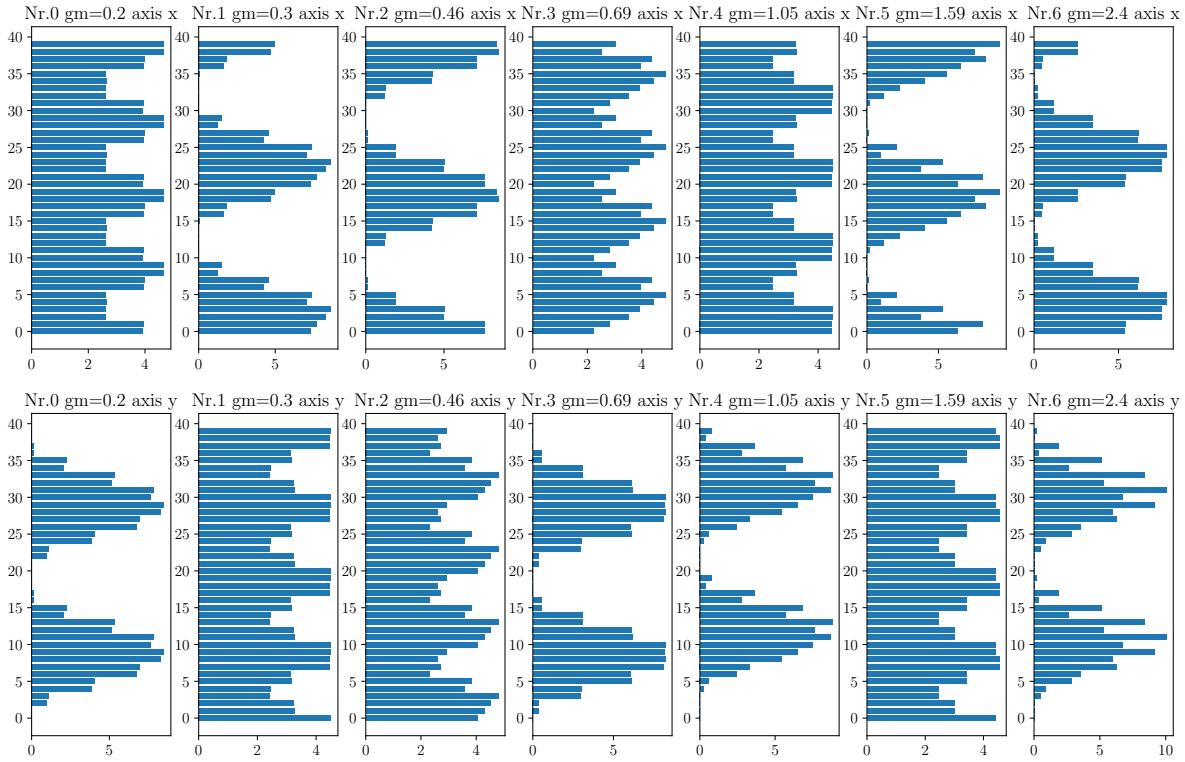
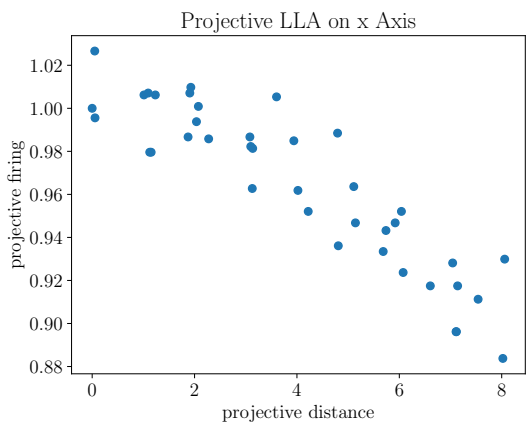
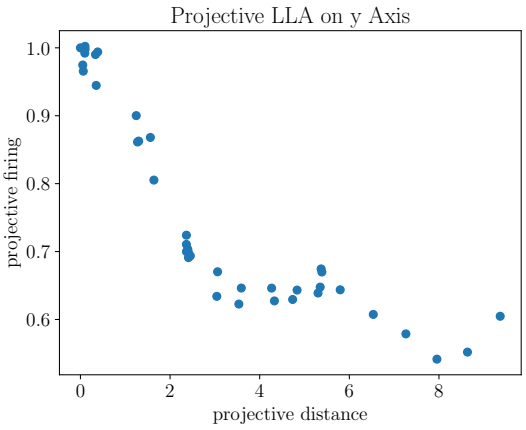


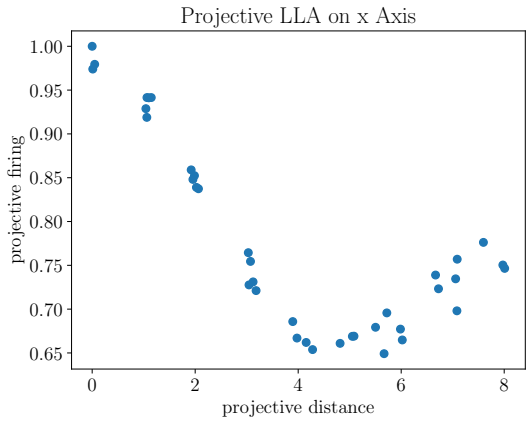
Figure 5.5: This figure shows the projective firing pattern of the grid cell system composed by me. On the x axis, four grid cell modules show clear spikes. On y axis, four modules show clear spikes. This is much more diverse than the original implementation in The baseline method shown in figure 5.2, where only two modules show clear spikes on x axis while five modules show clear spikes on y axis. The diversity of spiking characteristics has led to the result shown in figure 5.6.



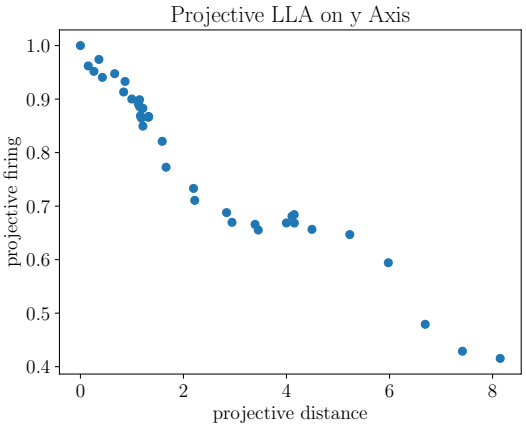
(a) the baseline method on the X axis.



(b) the baseline method on the Y axis.



(c) BioNav method on the X axis.



(d) BioNav method on the Y axis.

Figure 5.6: This figure shows the improvement after using a grid cell system that has alternating grid cell module directions. Figure 5.6a and figure 5.6b show the relation between the projective firing value and the projective distance using the baseline method. Figure 5.6c and figure 5.6d show the relation between the projective firing value and the projective distance using the BioNav method. The conclusion can be drawn, that the baseline method, whose grid cell modules show clear spikes only on y axis, can only show correlation between the projective firing value and the projective distance with accuracy along y axis. BioNav method, whose grid cell modules show clear spikes on both axes, shows correlation between the projective firing value and the projective distance with accuracy along both axes. It is worth noticing that the y axis in figure 5.6a starts from 0.80 while the y axis in figure 5.6c starts from 0.65. In this figure, on x axis, the positions where the projective firing value over 0.90 stays within 1.0m range in BioNav, and the projective firing value over 0.90 is around the range of 4.0m in the baseline method.

$$
\begin{aligned}
M =& \{m \mid m \text{ shows clear spikes in the current direction}\} \\
norm =& \sum_{m \in M} \frac{1}{g_m} \\
w_m =& g_m / norm
\end{aligned}
\tag{5.3}
$$

In this equation, m represents an individual grid cell module in the grid cell system. w_m represents the calculated weight of this grid cell module. The sum of all w_m equals 1. The w_m is larger when the g_m is smaller(represents a larger scale).

Using the weight calculated from above, the projective firing value becomes more monotonic according to the projective distance. The curve-like feature, where the projective firing values go up when the projective distance goes up, is corrected This is demonstrated in figure 5.8. This shows that our method, which calculates the weights of grid cell modules according to their g_m values, is more appropriate than the weight calculation in the baseline method.

5.4.4 Result after Improvement

Until now, we have made improvements by rearranging the grid cell modules to produce more diversity in the grid cell system, and adopting new weight calculation methods. The agent is able to carry out vector based navigation with enough accuracy. The lack of accuracy in the navigation method in the baseline method has been solved. The result is shown in figure 5.7.

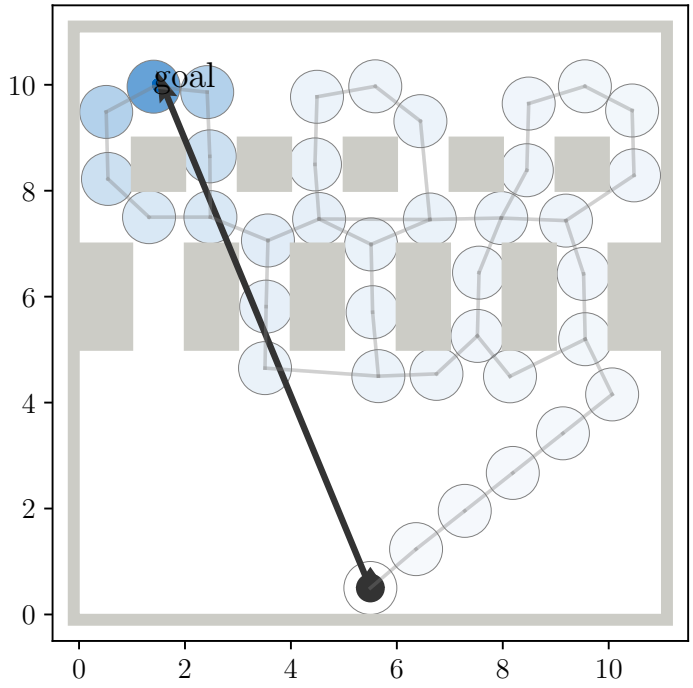
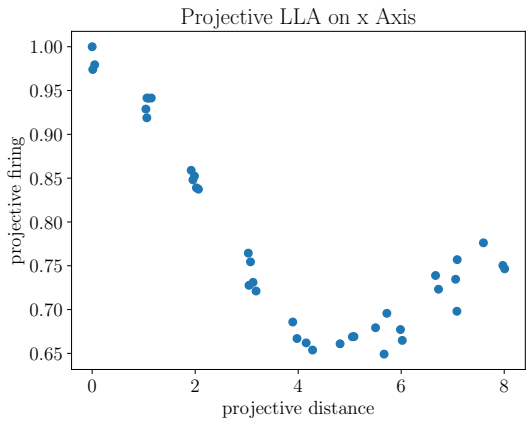
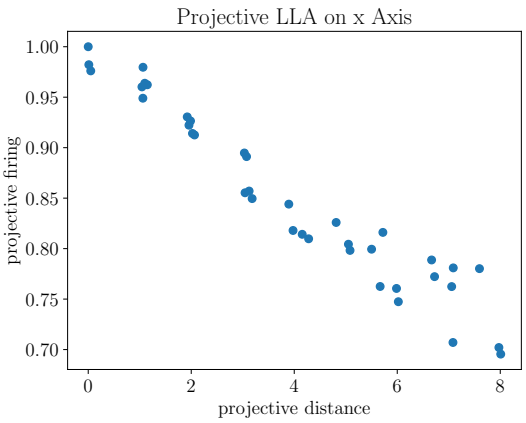


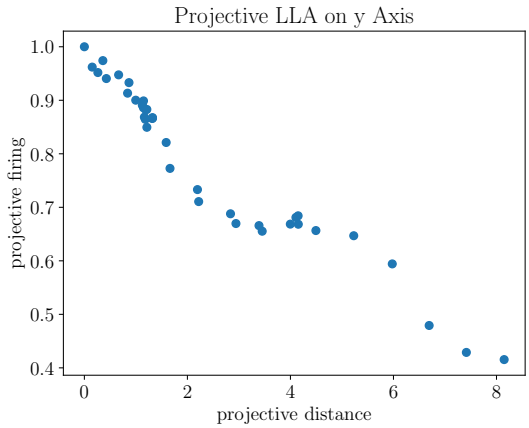
Figure 5.7: This figure shows the result after making the improvements. The agent is able to perform vector based navigation from the starting position to the goal position with accuracy.



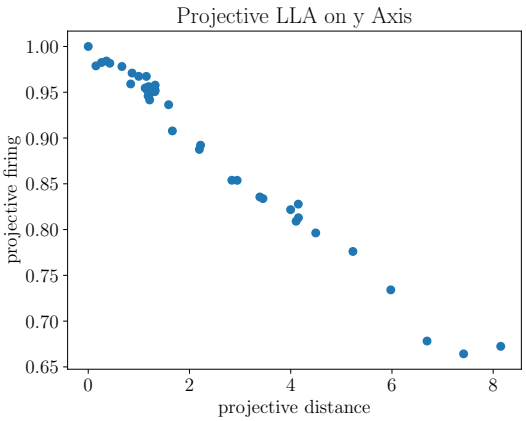
(a) Projective firing using the same weight on the x axis.



(b) Projective firing using the g_m relative weight on x axis.



(c) Projective firing using the same weight on y axis.



(d) Projective firing using the g_m relative weight on y axis.

Figure 5.8: This figure shows the difference between before and after using weight calculation according to the g_m values of grid cell modules. Figure 5.8a and 5.8c is the relation between projective firing value and projective distances before using g_m relative weight calculation. This is the same weight calculation method used in the baseline method. The relation between projective firing value and the projective distance is not monotonic using this weight calculation. Figure 5.8b and 5.8d show the relation between projective firing value and projective distances after using g_m relative weight calculation. This is the weight calculation method used in BioNav using a grid cell system. The relation between projective firing value and the projective distance is now monotonic. This shows that our way of weight calculation is more appropriate than the weight calculation method in the baseline method.

6 Time Consumption Problems and Speedup Attempts

6.1 Time Consumption Analysis

BioNav achieves its spatial cognition(path integration and navigation) through a simulation of a biological model. This simulation of the biological is achieved by updating many layers of the neural sheet in the grid cell modules. This approach has been proven to be effective. However, keeping biologically consistent with the current state of the agent is very costly. In current implementation, one update of the biological model consists of the following parts:

- 1. update the firing patterns of each neural sheet in the grid cell module(path integration): $(N_{gc} \cdot N_{gc})^2$ floating point calculations. $(N_{gc} \cdot N_{gc})^2 \cdot M$ for updating all neural sheets in the grid cell system.
- 2. check the firing values of the place cells: $N_{pc} \cdot (N_{gc} \cdot N_{gc}) \cdot M$ calculations.
- 3. derive useful information from prefrontal cortex(cognitive map): N_{pc} floating point calculations.

N_{gc} represents the size of the neural sheet in a grid cell module, while M represents the number of grid cell modules that make up a grid cell system. In the BioNav using grid cell system, we use $N_{gc} = 40$ and $M = 7$. N_{pc} represents the number of place cells in the constructed cognitive map. N_{pc} is dependent on the precision of the cognitive map and the scale of the environment. Thus the estimated time consumption of one update step is: $O(N_{gc}^4 \cdot M + N_{pc} \cdot N_{gc}^2 \cdot M)$.

Adding to this huge amount of computational consumption during the update, the update has to be done frequently enough, otherwise the model will become unstable[9]. As is stated in the paper of Edvardsen[9], the velocity inputs of the agent should be in the range from 0.1 m/s to 1.2 m/s, and the update time interval(dt) is advised to be set as 10ms($1 \cdot 10^{-2}s$). In BioNav, we set the average speed v to 0.5m/s and $dt = 10ms$. The covered length of each system update step is thus $dt \cdot v$.

To estimate the length of the trajectory during navigation. We quantify the complexity of the environment using α_{env}. The radius of the 2D environment by L_{env}. Thus the estimated number of update steps needed is: $O(\frac{\alpha_{env} \cdot L_{env}}{dt \cdot v})$.

Because the number of times the linear look ahead is relative to the complexity of the environment, it is hard to measure. We quantify it by multiplying α_{env} with a constant factor k_{LLA}, which is $k_{LLA}\alpha_{env}$. The time consumption of each linear look ahead can be estimated

as the radius of look ahead(R_{LLA}) divided by the covered length of each step. In another word, estimated time consumption of conducting one linear look ahead can be estimated as: $O(\frac{R_{LLA}}{dt \cdot v}) \cdot O(N_{gc}^4 \cdot M + N_{pc} \cdot N_{gc}^2 \cdot M)$.

Thus, the estimated time consumption of the BioNav using grid cell system navigating through an environment is:

$$[O(\frac{\alpha_{env} \cdot L_{env}}{dt \cdot v}) + k_{LLA}\alpha_{env} \cdot O(\frac{R_{LLA}}{dt \cdot v})] \cdot O(N_{gc}^4 \cdot M + N_{pc} \cdot N_{gc}^2 \cdot M) \tag{6.1}$$

After organizing according to factors:

$$O(\frac{\alpha_{env}}{dt \cdot v}) \cdot O(L_{env} + k_{LLA}R_{LLA}) \cdot O(N_{gc}^4 \cdot M + N_{pc} \cdot N_{gc}^2 \cdot M) \tag{6.2}$$

On the first sight, this might look complicated and incomparable to the conventional methods. However, it is worth mentioning that, even though the time consumption of BioNav seems to be deeply affected by $N_{gc} \cdot M$, this value is a set value and is not affected by the scale or complexity of the environment. Furthermore, the N_pc, which represents the number of place cells in the cognitive map, can be estimated as $O(L_{env}^2)$. Thus, if we only look at the dependency of the BioNav method to the environment, while modeling the set value(computation assumption by the system) as value K_{set}, the estimated time consumption of the BioNav becomes:

$$\alpha_{env} \cdot O(L_{env} + k_{LLA}R_{LLA}) \cdot O(L_{env}^2) \cdot K_{set} \tag{6.3}$$

This time consumption suddenly becomes a very good result, because it implies that the time consumption of BioNav only grows linearly when the problem becomes more difficult.

As a comparison, one of the most used path computational algorithms in conventional methods is A*. The time consumption of A* is $O(d^b)$. The d is the topological depth of the goal to the starting point, which can be estimated by $\alpha_{env} \cdot L_{env}$ and the b the branching factor in the graph. When calculating using the grid based conventional method, the branching factor is 4(4 neighbor cells in the topological graph). Thus the time consumption of A* using a obstacle map can be estimated as:

$$O(\alpha_{env}^4 \cdot L_{env}^4) \tag{6.4}$$

Comparing the two estimates of time consumption, the main advantage of BioNav against conventional methods in the category of time consumption is that BioNav can scale better in the "difficulty" of the problem than conventional methods. Here the "difficulty" of a problem can be defined as change in topological depth of the goal to the starting point when changes appear in the environment. This is closely related to α_{env}. As is shown in Equation 6.3, time consumption of BioNav only grows linearly according to α_{env} while A* shows polynomial growth in time consumption according to α_{env}. This can be further demonstrated in the comparison experiment between D* Lite and BioNav methods in chapter 9.

6.2 Possible Solutions

After analyzing the problems, there are two ways to solve the problem, that the current method consumes too much time:

- speeds up the computation without changing the model of the system.
- replace the part of the model that is too slow, with other alternative models that are much faster.

Here we will briefly describe our attempts using each of the methods.

6.2.1 Attempt Nr.1: Parallel Update of Grid Cells Using CUDA

To speed up the system without changing the model, one of the best ways is through running the system in parallel. For conventional methods like A* running in parallel is very hard to achieve because the computation is largely interdependent on each other. However, in biologically inspired methods, parallel computation is inherently achievable, because each neuron in the system can be viewed as an independent process. Using this character, we used the CUDA toolkit in Numba to achieve parallel programming using GPU, where each grid cell in the grid cell system is assigned to its only process.

The parallel system first calculates the distance between each pair of neurons according to its assigned heading direction in the system. Then the system calculate the synaptic weight between each pair of neurons according to the distance using the following equation:

$$rec(d) = e^{-\gamma d^2} - e^{-\beta d^2} \tag{6.5}$$

In this equation, $\gamma = 1.05 \cdot \beta$, $\beta = \frac{3}{\lambda^2}$ and $\lambda = 15$. These are the same parameters as used in Edvardsen's paper[9].

When performing update of grid cell system, each process solves the following equation using the calculated synaptic connection with other grid cells:

$$\tau \frac{ds_i}{dt} + s_i = f(s \cdot w^{rec}_{x_i - \hat{e_\theta}}) \tag{6.6}$$

Here the $\tau = 100$ms and $dt = 10$ms, s is the firing of another grid cell, $w^{rec}_{x_i - \hat{e_\theta}}$ is the synaptic weight from this grid cell calculated according the the above mentioned calculation method. This method of update calculation is described in the paper of Edvardsen[9], and will not be covered in detail here.

Using the parallel update method, the update speed of the system is greatly increased. Figure 6.1 shows an experiment, where the original linear implementation and the CUDA implementation(parallel implementation) both perform the same 10,000 update steps. The CUDA implementation is 4 times quicker than the linear implementation(using Numpy array, not fully parallel). However, this speedup is only possible if the grid cell system is the only model being measured. If the CUDA grid cell model, which runs on GPU, is embedded to the whole system of BioNav, the overhead begins to appear, because the place cell system

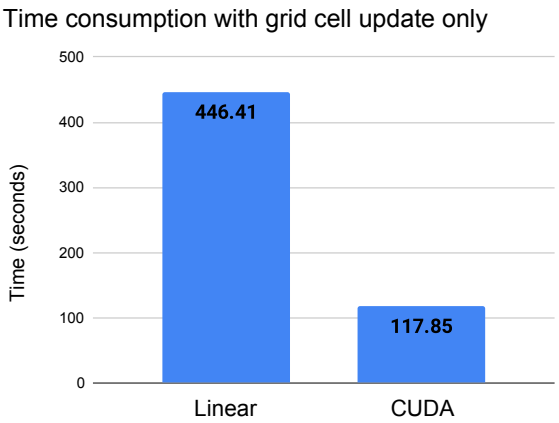
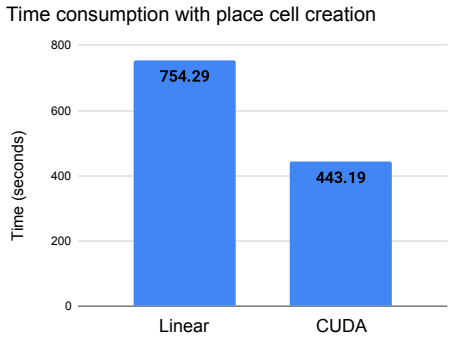


Figure 6.1: Time consumption comparison only with grid cell system. This measurement is taken by only updating the grid cell system by 10,000 update steps. The parallel system on GPU performs much better than the linear system on CPU. The speed of the parallel system is around four times faster than the linear system(the linear system uses Numpy array during calculation, and thus can also be considered as incomplete parallel computation). This experiment shows that the grid cell system can have significantly better performance using parallel computation. This also shows that BioNav has the potential to have better performance using parallel methods. Moreover, this implies that biologically inspired navigation methods, like BioNav, can gain benefit from parallel computation, which is very hard for conventional methods.

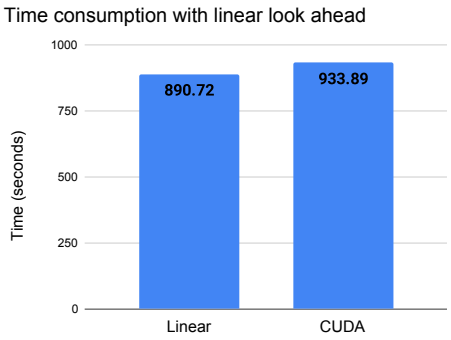
needs to copy the firing pattern of the grid cell system to the CPU from GPU. As is shown in figure 6.2, the CUDA system only performs one time quicker than the linear implementation if we include creation of place cells into the system. The CUDA system is even slightly slower than the linear implementation, if we include linear look ahead into the system, where the firing pattern of grid cell system in the GPU needs to be temporarily stored in CPU during look ahead virtual updates and uploaded to the GPU again after look ahead finishes.

Despite the disappointing result received after embedding the parallel grid cell system into the whole system, it is enough to prove that a single biologically inspired system can be sped up significantly using parallel computation. This alone shows the potential of biologically inspired navigation methods to achieve much faster speed using parallel methods, while it is very hard for conventional methods to achieve faster speed through parallel methods.

It is also reasonable to believe that, if the whole system of BioNav is achieved using a parallel method, the performance can be significantly boosted. However, considering the initiative of this thesis, making the whole system parallel requires too much effort for a single statement. Thus, the parallel implementation of the whole BioNav method is beyond the scope of this thesis.



(a) Time consumption comparison with place cell creation in the system.



(b) Time consumption comparison with linear look ahead.

Figure 6.2: This figure shows that the benefit of CUDA parallel computation is diluted by overhead of GPU and CPU communication, if more systems are included in the update. Figure a) shows the comparison result between linear and parallel if place cell creation is included. This measurement is done by comparing the time consumption of exploring the maze, where the grid cell system is not only updated, but also passes its firing values from GPU to CPU to create place cells using a snapshot of grid cell system firing patterns. The speed of the parallel system is now only one time faster than the linear system. Figure b) shows that the benefits of using parallel computation on GPU is outweighed by the GPU CPU overhead, if linear look ahead is also included in the system. This measurement is taken from the navigation through the maze to the goal. During navigation, the firing patterns of grid cell systems are not only passed from GPU to CPU to compute place cell firing value, but also passed from CPU to GPU to reset the grid cell firing values during linear look ahead. The speed of the parallel system is even slightly slower than the linear system because of the overhead

6.2.2 Attempt Nr.2: Using a More Efficient Model than Grid Cell System

The second way to solve the time consumption problem is using a simpler model to replace the time consuming model.

As is shown in Equation 6.2, the update of grid cells has put a very big burden on the system. If there is an alternative model which requires much less computation while achieving similar functionality of grid cell systems, then we could solve the problem of time consumption by reducing the K_{set} in the Equation 6.3.

After a careful examination of the grid cell system, we found that it is possible to replace the grid cell system using a coordinate system model. The path integration functionality of the grid cell model is now achieved by keeping an allocentric coordinate position to the initial start position, and adding the speed signal to this position. A more detailed description of replacing the grid cell system with a coordinate system can be found in chapter 7.

Using the coordinate system to replace grid cell model, the update of BioNav using coordinate system also consists of the following steps:

- 1. update the current allocentric coordinate using the speed signal(path integration): $O(1)$ floating point calculations.
- 2. check the firing values of the place cells: $O(N_{pc})$ calculations.
- 3. derive useful information from prefrontal cortex(cognitive map): $O(N_{pc})$ floating point calculations.

This makes the time consumption of the BioNav using coordinate system model:

$$O(\frac{\alpha_{env}}{dt \cdot v}) \cdot O(L_{env} + k_{LLA} R_{LLA}) \cdot O(N_{pc}) \tag{6.7}$$

Thus, replacing the grid cell system with a coordinate system makes the BioNav method significantly faster, while achieving similar functionality. This can be further demonstrated in chapter 9.

7 Simplification using Coordinate System Model

7.1 Meaning of Simplification

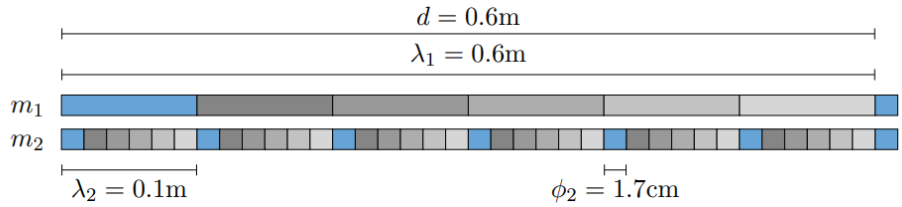
The BioNav model was first proposed to solve the problem of robotic navigation in a better way, using the discoveries in neuroscience. Thus, the grid cell system, the place cell system and the prefrontal cortex system are created in a way to simulate the activity patterns in a rodent's hippocampus, hoping that the model could be able to achieve similar navigational abilities with rodents' brains by using similar compositions. However, these simulations require a very large amount of calculations, which means that it is very sophisticated and slow. However, is this the complexity of calculating the grid cell system needed to achieve the functionalities?

After playing with the BioNav model intensively, we found that the allocentric coordinates system in the 2D plane is a potential replacement to the grid cell system(the BioNav using coordinate system will be referred to as Coordinate System BioNav, while the BioNav using grid cells will be referred to as Grid Cell BioNav). The explanations of this judgment will be given in the next section.

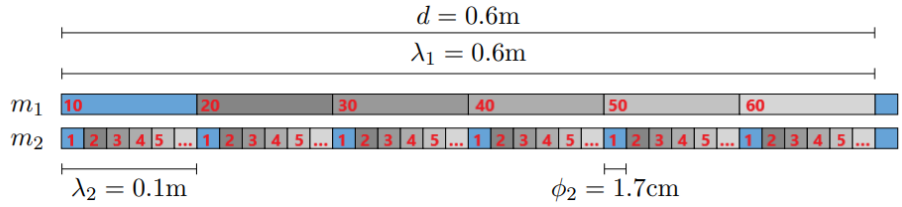
The Coordinate System BioNav replaces only the grid cell system and the interface between grid cells and place cells with a coordinate system, while leaving other parts of the methods unchanged including the structure and workflow of the method. It functions as a simplification of Grid Cell BioNav, there is not yet theoretical proof that Coordinate System BioNav is able to fully replace the properties of Grid Cell BioNav except some experiment in the maze setup proving that the Coordinate System BioNav behave very similarly in the maze and achieves the functionalities promised by Grid Cell BioNav. However, because Coordinate System BioNav also uses a generated cognitive map consisting of prefrontal cortex cells, relies on the firing of place cells to navigate in the environment, and uses Linear Look Ahead to make decisions during navigation, Coordinate System could also be categorized into biologically inspired navigation methods.

7.2 Replacing Grid Cell System with Coordinate System

In the paper of Edvardson 2015 proves that grid cell systems can achieve path integration in 2D space[9]. In the model of BioNav, path integration is the functionality that the grid cell system provides to the whole system. Path integration of the grid cell system is achieved through embedding neural sheets of different update scales together, so that a location in the



(a) Nested approach of grid cell system.



(b) Similarities between nested approach and the composition of numbers.

Figure 7.1: This figure shows the similarity in the logic of the nested approach of the grid cell system and the composition of numbers. Figure 7.1a shows the composition of grid cell systems using grid cell modules with different scales in one dimension. Figure 7.1b shows that the logic of the composition of numbers and the nested approach is similar in one dimension.

environment can be expressed uniquely by the combined firing pattern of all neural sheets. Two ways were proposed in the paper: a nested approach and a combinatorial approach, which is demonstrated in figure 2.2. These two approaches all use grid cell modules of different scales to express positions in a 2D space, which makes them fundamentally the same. As shown in figure 7.1, the logic of nested approach and the logic of numbers using different digits to represent values of different scales are very similar to each other. This similarity can be further expanded into 2D space, where the grid cell system shares the similar logic with a coordinate system.

To replace the grid cell system with a coordinate system, we let the agent store its starting position as $(0.0, 0.0)$, which corresponds to the storage of initialization values of the grid cell system. The agent performs path integration on the coordinate system by summing its initial coordinate and the speed signals together. Performing path integration on a coordinate system replaces the path integrating functionality of the grid cell system. The firing patterns of the grid cell system, which represents the current position of the agent, is now replaced by an allocentric coordinate value, which is calculated by summing all the speed signals until the current moment together.

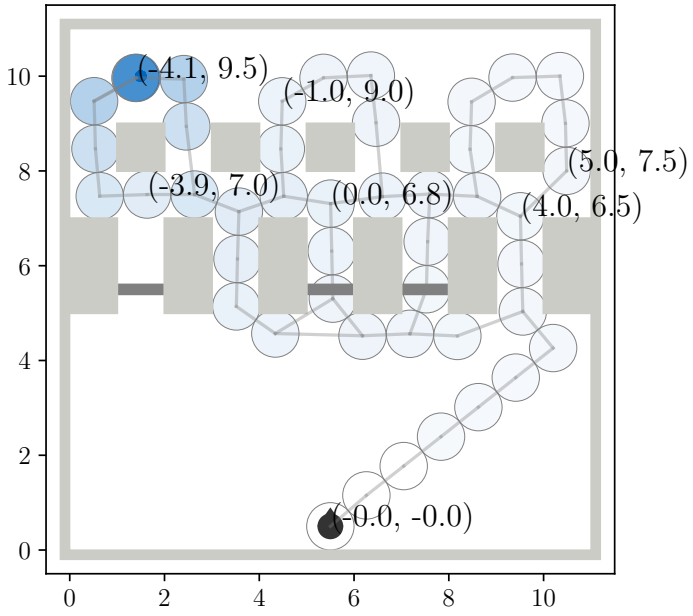


Figure 7.2: This figure shows the created cognitive map with allocentric coordinates. The cognitive map is created using threshold= 0.5, max_range= 5.0 and firing_threshold= 0.85. As is shown in the figure, the place cells are represented by its allocentric coordinate, with (0,0) at the starting position. The place cell representing the goal has the coordinate of (-4.1,9.5).

7.3 Place Cells

7.3.1 Representing the Position

Place cells, which provide the agent the awareness of places it has been to before, was represented by a snapshot of the firing patterns of the grid cell system. Because as is shown in section 7.2, the firing patterns can be replaced by an allocentric coordinate. Thus, the place cell records the current allocentric coordinate.

7.3.2 Firing Value

The firing value of the place cells becomes higher when the firing pattern of the grid cell becomes more similar to the snapshot stored in the place cell. In another word, the firing value of a place cell increases when the agent approaches the allocentric position it represents.

To mimic these characteristics, we have designed a firing value function which is characterized by three variables: threshold(t), max_range(r_{max}) and firing threshold(f_{thres}). The firing value is calculated using the following equation, with *dis* representing the allocentric distance between current allocentric coordinate of the agent and the recorded allocentric coordinate by the place cell p:

$$firing_p = \begin{cases} \frac{(t - dis) \cdot (1.0 - f_{thres})}{} + f_{thres} & dis \leq t \\ \frac{(r_{max} - dis) \cdot f_{thres}}{(r_max - t)} & t < dis \leq r_{max} \\ 0 & otherwise \end{cases} \tag{7.1}$$

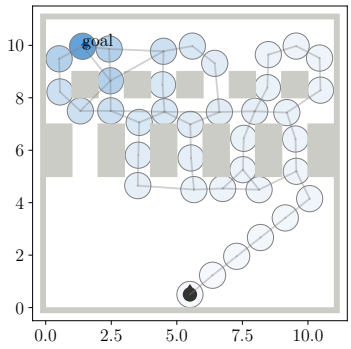
The projective firing value is computed with the same equation using the projective distance between two coordinates instead.

7.4 Other Modules in the System

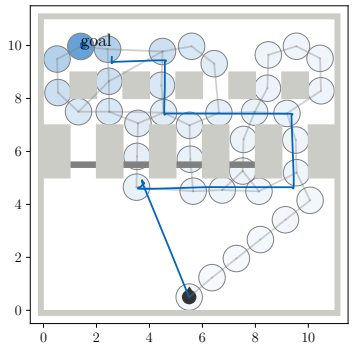
The other modules including the exploration, navigation, linear look ahead are all the same.

7.5 Results

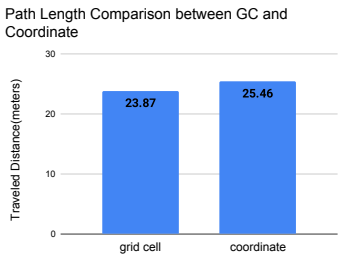
Figure 7.3 shows that the BioNav using coordinate system is similar in functionality to the BioNav using grid cell system. As is shown in the figure, using the same exploration trajectory, the two methods create a similar cognitive map and both can navigate through the maze with a changed gate setup. There will be more examples demonstrating the functional similarities in chapter 9. In addition to the similarity in functionality, the BioNav using coordinate system is much better than the BioNav using grid cell system in time consumption. In the same experiment setup, where all the gates are open during exploration and only gate Nr.5 is open during navigation, the BioNav using coordinate system took only 12 seconds while the BioNav using grid cell system takes over 2000 seconds.



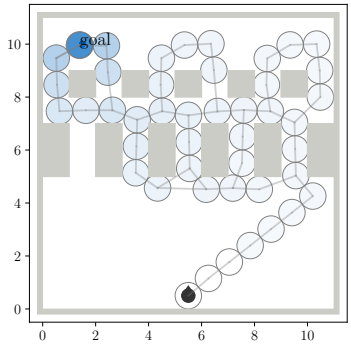
(a) The cognitive map using grid cell system.



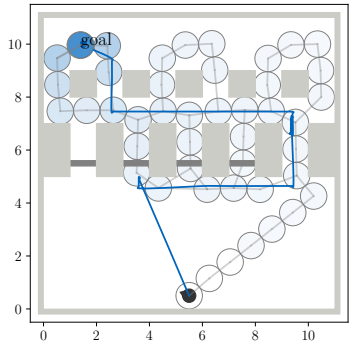
(b) The navigation trajectory using grid cell system.



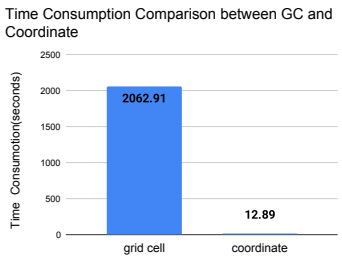
(c) Comparison of traveled distance.



(d) The cognitive map using coordinate system.



(e) The navigation trajectory using coordinate system.



(f) Comparison of time consumption.

Figure 7.3: This figure compares the BioNav using a coordinate system with the BioNav using a grid cell system. As is shown in figure a) and d), the two methods construct a similar cognitive map following the same trajectory during exploration of the maze. As is shown in figure b) and e), the navigation trajectories through the maze to the goal of the two methods in the same maze set up are similar to each other. Figure a), b), d), and e) shows that the coordinate system could be a valid replacement of the grid cell system. Figure c) shows the comparison of the traveled distance of the two methods during navigation. The trajectory of the BioNav using a grid cell system is around 5% shorter than the trajectory of the BioNav using a coordinate system. However, the figure f) shows that the BioNav using coordinate system is around 99% faster than the BioNav using grid cell system. Figure c) and f) show that the BioNav has a huge advantage in time consumption while sacrificing a small margin of its traveled distance. This is a strong statement that the BioNav using a coordinate system is better than the BioNav using a grid cell system.

8 Conventional Methods and D* Lite

There are many conventional methods that solve the problem of navigation in a dynamic environment. These approaches include: sampling based methods and grid based methods.

8.1 Sampling Based Method: Probabilistic Roadmap

The sampling based methods can be represented by Probabilistic Roadmap Path Planning(PRM). A roadmap is a graph with interconnected nodes in the environment, where all edges are not blocked by any obstacles. The start position and goal position are also treated as nodes in the roadmap. An example of a construction of a roadmap is demonstrated in Figure 8.1.

The agent can compute a path from start position to its goal by simply using a path finding algorithm like Dijkstra. In Probabilistic Roadmap Path Planning(PRM), an agent first needs to create an obstacle map of the environment, and compute the feasible space in the environment(feasible space means the areas that are accessible to the agent, despite constraints, like the volume of the robot.). The agent then generates nodes of the road map through random sampling in the accessible space. Then the agent connects the generated nodes if the vectors between them are also in the feasible space. If the roadmap does not contain any path to the goal, the agent regenerates the roadmap with more nodes until a path leading to the goal exists in the roadmap. PRM has a solution probability of 1, which means if a path leading to the goal exists, PRM will eventually generate one feasible path. However, despite its strong capabilities for solving the problems, there are many weaknesses. One of the weaknesses is the computation time of the problem. During each generation of a roadmap, the same number of Boundary Value Problems as the number of potential edges need to be solved. This is a big burden for computation. Another weakness is that, it is difficult to achieve optimal path through the environment because of the random generation of roadmap in this problem.

In the problem setup of this paper, grid based methods are more superior than sampling based methods. This is because the environment in this paper is a maze on a 2D plane. The agent is free to move along two axes of the plane. Grid based methods have a more detailed roadmap and quicker solution to the boundary problem because every cell in the grid map implies a node in the roadmap, and only neighboring non-obstacle cells are connected to each other.

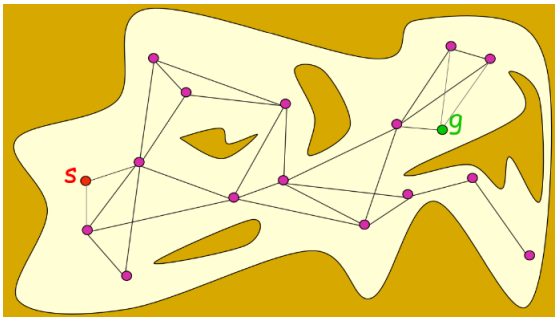


Figure 8.1: This figure shows the construction of a roadmap in an environment. The feasible space is marked by yellow, while the infeasible space and obstacles are marked by brown. The red s denotes the start position of the agent and the green g denotes the goal. The purple nodes are nodes that are generated randomly in the feasible space. Two nodes are connected if there are no obstacles between them and the distance between them is within a threshold. As is shown in the figure, there exists a path from start position to the goal in the generated roadmap.

8.2 Grid Based Method: from Static Dijkstra to Dynamic D* Lite

An exact explanations of the D* Lite algorithms are tedious. This section is only a rough explanation and only covers the most important concepts in D* Lite. For a more detailed explanation, please refer to the paper by Sven Koenig[22]. The implementation of algorithms is in python and uses part of the source code by the implementation from Matt Deyo[24].

Grid based methods are closely related to graph based methods. The most well known graph based method is the Dijkstra Algorithms. Dijkstra algorithms take in a connected graph with costs to each edge. It computes the path with the lowest cost from one point the another point in the graph by recurrently exploring the unexplored node that has the lowest cost. Grid based method uses a grid map instead of a connected graph. A 2D grid map is constructed by subdividing the feasible space into a set of cells which are only connected to its geometrical neighbors. Then the Dijkstra algorithm is applied on the constructed 2D grid map, where the cells are considered nodes.

However, the graph used by Dijkstra algorithms is considered static. Thus if changes in the environment appear, for example some new obstacles blocking the original path appear, the algorithms should be able to generate new paths based on the changes. To deal with changes in the map, Life Long Planning A* algorithms were proposed. A* algorithms is a variant based on Dijkstra's algorithms, where it introduces a heuristic value, for example the distance between the current node to the goal. The heuristic value is a hint to guide the search to be performed in a more efficient way. The heuristic values need to fulfill the triangle inequality: $h(s_{goal}, s_{goal}) = 0$ and $h(s, s_{goal}) \leq c(s, s') + h(s', s_{goal})$ for all nodes $s \in S$ and $s' \in Succ(s)$ with $s \neq s_{goal}$, where $c(s, s')$ is the actual cost from node s to node s', and $Succ(s)$ is the set of all nodes that are connected by node s by an edge(successor).

The Lifelong Planning A* introduces a new variable *rhs* which is short for "right hand

procedure CalculateKey(s)
{01} return $[\min(g(s), rhs(s)) + h(s, s_{goal}); \min(g(s), rhs(s))]$;

procedure Initialize$()$
{02} $U = \emptyset$;
{03} for all $s \in S$ $rhs(s) = g(s) = \infty$;
{04} $rhs(s_{start}) = 0$;
{05} U.Insert(s_{start}, CalculateKey(s_{start}));

procedure UpdateVertex(u)
{06} if $(u \neq s_{start})$ $rhs(u) = \min_{s' \in Pred(u)}(g(s') + c(s', u))$;
{07} if $(u \in U)$ U.Remove(u);
{08} if $(g(u) \neq rhs(u))$ U.Insert(u, CalculateKey(u));

procedure ComputeShortestPath$()$
{09} while (U.TopKey()$\dot{<}$CalculateKey(s_{goal}) OR $rhs(s_{goal}) \neq g(s_{goal})$)
{10} u = U.Pop();
{11} if $(g(u) > rhs(u))$
{12} $g(u) = rhs(u)$;
{13} for all $s \in Succ(u)$ UpdateVertex(s);
{14} else
{15} $g(u) = \infty$;
{16} for all $s \in Succ(u) \cup \{u\}$ UpdateVertex(s);

procedure Main$()$
{17} Initialize();
{18} forever
{19} ComputeShortestPath();
{20} Wait for changes in edge costs;
{21} for all directed edges (u, v) with changed edge costs
{22} Update the edge cost $c(u, v)$;
{23} UpdateVertex(v);

Figure 8.2: Algorithms of Lifelong Planning A*.[22]

side". The variable is used to show inconsistency of a node after a change in the map appears. Lifelong Planning A* keeps three invariant during its life time:

- First invariant:

$$rhs(s) = \begin{cases} 0 & \text{if } s = s_{start} \\ \min_{s' \in pred(s)} g(s') + c(s', s) & \text{otherwise} \end{cases} \tag{8.1}$$

- Second invariant: A node s is inconsistent if $g(s) \neq rhs(s)$. Lifelong Planning A* keeps a priority queue U consisting of all inconsistent nodes.

- Third invariant: In the priority queue U, the key of a node is a two value bracket: $[k_1(s); k_2(s)]$, where $k_1(s) = \min(g(s), rhs(s)) + h(s, s_{goal})$ and $k_2(s) = min(g(s), rhs(s))$. Node s has higher priority in the queue if $k_1(s) \leq k_1(s')$ or $(k_1(s) = k_1(s')$ and $k_2(s) \leq k_2(s'))$.

Figure 8.2 shows the algorithms of Lifelong Planning A*. In the first iteration, the algorithms computes the optimal path to s_{goal} from s_{start} using the same method with A*. The algorithms is the same as A* in the first phase because it calculates the path by expanding the nodes with the highest priority(sum of $g(s)$ and $h(s)$, same in A*) is the priority queue U until there are no nodes in U that have a lower priority value than s_{goal}(no potential shorter path is present). Then the algorithm waits for the changes to appear. If the cost between the node u and v is changed from $c(u, v)$ to $c'(u, v)$, the *rhs* value of the node v is updated and node v is added to the priority queue U because node v is now inconsistent($g(v) \neq rhs(v)$). Then the algorithm performs a $ComputeShortestPath()$, where node v and all the nodes that were using node v in their path are added to the priority queue U and recalculated. This update is similar to A*, but only the "necessary" nodes are recalculated. The update of a node is "necessary" if the node is in the priority queue U and its key value is smaller than the key of s_{goal}. In this way, the update of nodes are kept local to the area that are affected by the changed edge and a global recalculation using A* can be avoided.

However, Lifelong Planning A* is not applicable to the scenarios in robotics, where the robot moves to the next location between updates and the changes in the environment are discovered only when they are in sight of the robot. This creates problems, firstly because a moving robot implies that s_{start} is constantly changing and makes the calculation of $g(s)$ unstable. Secondly, the incremental discovery of the changes implies the priority queue U needs to be reorganized very frequently, and thus requires optimization.

Figure 8.3 shows the algorithms of D* Lite. The difference between D* Lite and Lifelong Planning A* is as following:

- D* Lite searches from s_{goal} to s_{start}(LPA* searches from s_{start} to s_{goal}) and thus its g-values are estimates of the goal distances.
- A variable keeping track of the $h(s_{last}, s_{start})$ is added to provide a raising lower bound of the key values.

Searching from s_{goal} to s_{start} solves the problem of unstable data structure caused by movement, because the s_{goal} does not move and creates a stable basis for the calculation of $g(s)$. Furthermore, the nodes that are covered by the robot are usually not updated again because it has a higher key value than the $s_{current}$(current node of the robot). Keeping a lower bound k_m for the key value makes reorganizing the priority queue U less costly as the robot moves closer to its goal.

Figure 8.4 shows the common implementation of D* Lite through an intuitive example. In the common implementation of D* Lite, the feasible space of the robot is divided into an eight-connected grid. The obstacles are represented as cells that are blocked(painted dark) in the grid. A robot has a range of sight and can only detect changes that appear within this range. The knowledge of the environment is represented by the current map of the environment and a default path(shortest path using current map).In the example, the robot has the knowledge of the environment as the upper graph. However, after the robot has moved, it finds that unexpected obstacles have appeared which blocks its default path. It

procedure CalculateKey(s)
{01'} return $[\min(g(s), rhs(s)) + h(s_{start}, s) + k_m; \min(g(s), rhs(s))]$;
procedure Initialize()
{02'} $U = \emptyset$;
{03'} $k_m = 0$;
{04'} for all $s \in S\ rhs(s) = g(s) = \infty$;
{05'} $rhs(s_{goal}) = 0$;
{06'} U.Insert$(s_{goal}$, CalculateKey$(s_{goal}))$;
procedure UpdateVertex(u)
{07'} if $(u \neq s_{goal})\ rhs(u) = \min_{s' \in Succ(u)}(c(u, s') + g(s'))$;
{08'} if $(u \in U)$ U.Remove(u);
{09'} if $(g(u) \neq rhs(u))$ U.Insert$(u$, CalculateKey$(u))$;
procedure ComputeShortestPath()
{10'} while (U.TopKey()$\dot{<}$CalculateKey(s_{start}) OR $rhs(s_{start}) \neq g(s_{start}))$
{11'} $k_{old} =$ U.TopKey();
{12'} $u =$ U.Pop();
{13'} if $(k_{old} \dot{<}$CalculateKey$(u))$
{14'} U.Insert$(u$, CalculateKey$(u))$;
{15'} else if $(g(u) > rhs(u))$
{16'} $g(u) = rhs(u)$;
{17'} for all $s \in Pred(u)$ UpdateVertex(s);
{18'} else
{19'} $g(u) = \infty$;
{20'} for all $s \in Pred(u) \cup \{u\}$ UpdateVertex(s);
procedure Main()
{21'} $s_{last} = s_{start}$;
{22'} Initialize();
{23'} ComputeShortestPath();
{24'} while $(s_{start} \neq s_{goal})$
{25'} /* if $(g(s_{start}) = \infty)$ then there is no known path */
{26'} $s_{start} = \arg\min_{s' \in Succ(s_{start})}(c(s_{start}, s') + g(s'))$;
{27'} Move to s_{start};
{28'} Scan graph for changed edge costs;
{29'} if any edge costs changed
{30'} $k_m = k_m + h(s_{last}, s_{start})$;
{31'} $s_{last} = s_{start}$;
{32'} for all directed edges (u, v) with changed edge costs
{33'} Update the edge cost $c(u, v)$;
{34'} UpdateVertex(u);
{35'} ComputeShortestPath();

Figure 8.3: The algorithms of D* Lite.[22]

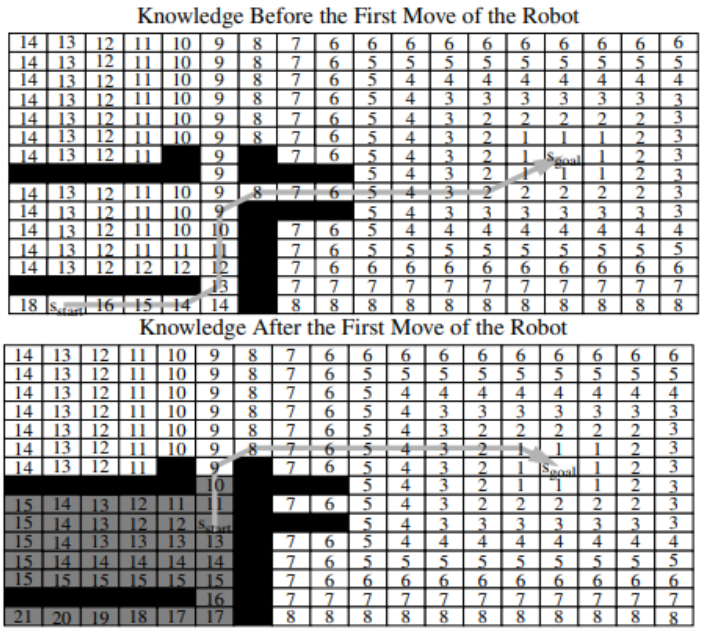


Figure 8.4: An example of D* Lite.[22]

updates its map, updates the necessary nodes and recalculates a path to its goal, which is shown in the graph underneath.

9 Comparison Experiments

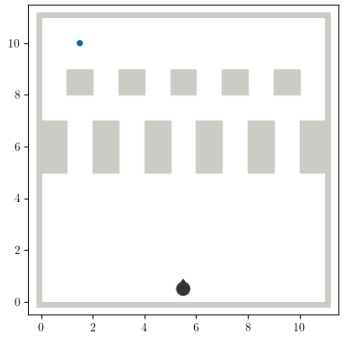
To show what features are provided by BioNav(our method) that the conventional methods can not provide, and also to show possible drawbacks of BioNav, we have set up comparison experiments between BioNav using grid cell system, BioNav using coordinate system, and D* Lite.

9.1 Choice of Conventional Method as Comparison

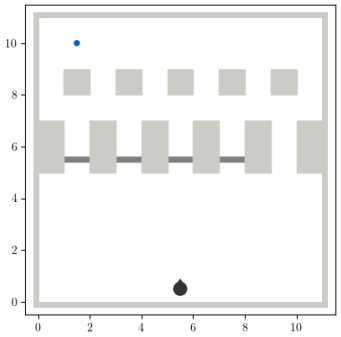
We chose D* Lite to represent the conventional methods to compare with BioNav, because D* Lite is a well established conventional method that navigates in dynamic environments. This matches the assumption of the environment of BioNav. D* Lite also allows the agent to use previous knowledge in an environment, which is a precomputed distance grid. This feature matches the explore-and navigate set-up of BioNav very well. Furthermore, the ability of D* Lite to find shortcuts is strong. This creates good competition to BioNav, one of whose advantages is finding shortcuts. Thus D* Lite functions as a good comparison for BioNav.
There are two ways of implementing D* Lite: one using a grid map and the other using a node graph. The experiment will focus mainly on the D* Lite using a grid map, because even though D* Lite using node graph is more simplified in representation and therefore faster than D* Lite using grid map, D* Lite using node graph does not have strong ability on finding shortcuts, which is a huge disadvantage when compared to BioNav. However, the D* Lite using grid map requires the agent to establish a detailed map of the environment, from which a grip map is extracted. This implies that the agent should not only have awareness of its own path during the exploration but also has the ability to acquire knowledge of its surroundings using a sensor, for example a LiDAR sensor. This is not required by D* Lite using node graphs, where node graphs can be established by calculating the choke-points on the exploration trajectory.

9.2 The Maze Set-up

The maze is as shown in the figure 9.1. The maze is surrounded by four walls. The maze can be split up into three parts. The lower part of the maze is an empty plane, which tests the behavior of the robot on empty planes, where shortcuts are presented because of the absence of obstacles. The middle part of the maze consists of five passages, each of them can be blocked with a door. The passages and doors scenario is very common in real-life and represents dynamic changes in the environment. This tests the ability of a robot to deal with the dynamic changes that are inconsistent with its previous knowledge. The upper part of



(a) The maze with all gates open, default setup during exploration.



(b) The maze with gate 1,2,3,4 closed and gate 5 open.

Figure 9.1: This figure shows the setup of the maze. The gray areas demonstrate the obstacles and walls, the dark gray shows the gates. The gates can be open or closed, which makes the environment dynamic. This maze setup captures a typical problem in real life, where there are several possible routes leading to a space with inner structures. The possible routes are passages consisting of a gate and a pathway. The inner structures are boxes behind the pathways. The blue dot in the maze marks the goal position, the black arrow marks the start position of the agent.

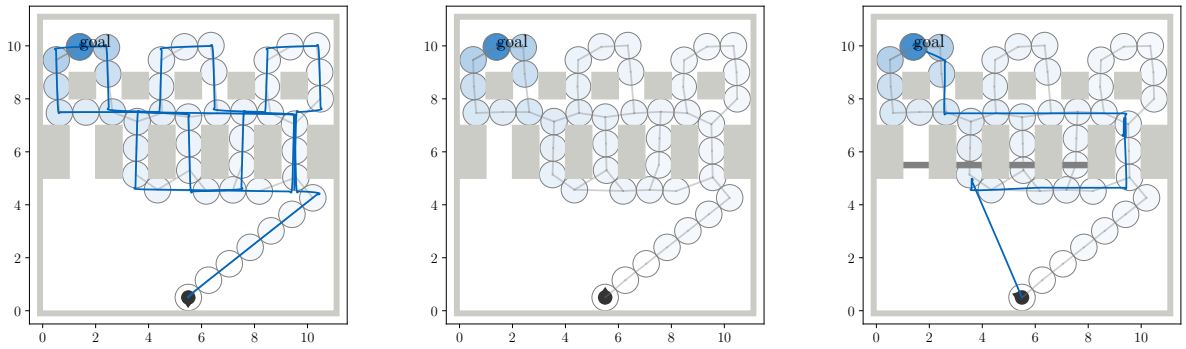
the maze consists of several columns. Between each pair of columns, there is a narrow path. This tests the ability of robot to navigate through static obstacles.
The goal is on the upper left part of the maze, to get to the goal, a robot needs to go first through the empty plane, and then find an open door, and then go through the passage, and then navigate through the columns, and at last, reach the goal.

9.3 Scenario

A maze described in the previous section is set as an environment. A robot first has a chance to go from its starting location to explore the maze. It discovers the goal during the exploration. After the exploration, the goal location remains the same, but the maze changes the set-up of doors, of which the robot is not aware. Then the robot goes from the same starting location and navigates through the maze to find the goal.

9.4 Acquisition of Knowledge about the Environment

Both of the agents using D*Lite and BioNav has a predefined path through the environment during exploration.



(a) The exploration trajectory of the agent.

(b) The constructed cognitive map of the agent.

(c) The navigation trajectory of the agent using BioNav.

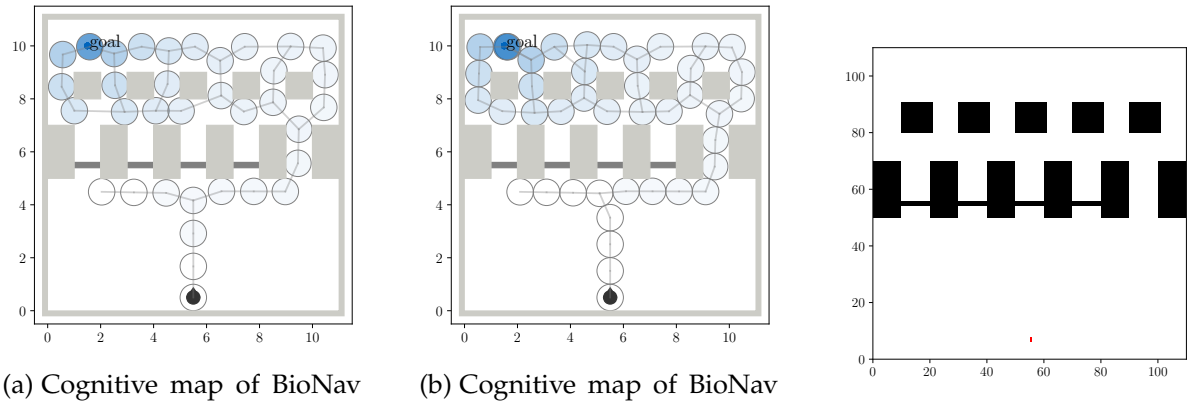
Figure 9.2: This figure shows the agent using BioNav first constructing a cognitive map during exploration of the maze, then the agent starts from the same start position and uses the cognitive map to navigate through the changed maze and reach the goal position. Figure 9.2a shows the trajectory that the agent followed during exploration of the maze. Figure 9.2b shows the constructed cognitive map of the agent following this trajectory. Figure 9.2c shows the trajectory of the agent using BioNav navigating through the maze, which has a changed setup of the gates from exploration of the maze, and reaching the goal position using the guidance from the cognitive map.

The agent using D* Lite reconstructs the part of the environment it has explored using a special sensor, and extracts a 2D obstacle grid map of the environment. the extracted 2D grid map functions as knowledge about the environment for the agent using D* Lite.
The agent using BioNav first initializes its modules, then it goes through the predefined path, and builds up a cognitive map using a record of its temporal speed and orientation. The cognitive map and the initialized modules function as knowledge about the environment for the agent using BioNav.

9.5 Strength of BioNav Versus D* Lite

9.5.1 BioNav Is More Powerful on Discovering Possible Shortcuts than D* Lite

Environment Setup. As is shown in figure 9.3.
D*Lite agent has a full reconstruction of the environment
Gate 5 is open, other gates are closed during exploration.
Gates 2 and 5 are open, other gates are closed during navigation.
Results of the experiment. The results are shown in figure 9.4. The agent using D* Lite finds the path through gate 5 with very quick decision time, while ignoring the short-cut through gate 2.
The agent using BioNav finds the path to the goal through the shortcut through gate 2. The



(a) Cognitive map of BioNav using grid cell system.

(b) Cognitive map of BioNav using coordinate system.

(c) Obstacle map of D* Lite.

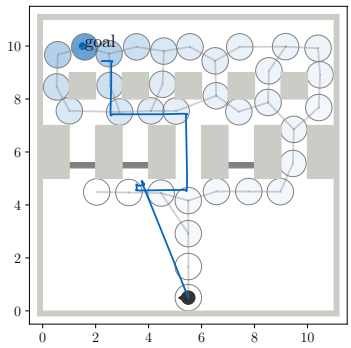
Figure 9.3: This figure shows the knowledge of the agents about the maze.

decision time is very long for BioNav using grid cells. The decision time is comparatively much shorter for BioNav using the coordinate system. But both versions of the implementation of BioNav travel through similar routes(finds path through gate 2). There are two versions of D* Lite using different precision of obstacle map construction. D*Lite 100 uses map construction of 100×100 cells, while D*Lite 50 uses the map construction of 50×50 cells. The D*Lite using a lower precision is considerably faster than the higher precision one.
Analysis of the results. BioNav is more flexible on dealing with changes, because it does not pre-assume obstacles in the environment, this keeps all possibilities open during navigation. This is because: In the cognitive map, there are only representations of possible passages, representations of obstacles are not present in the cognitive map. The agent first performs vector based navigation, this will provide higher possibilities to utilize other possible routes in the environment.
On the other hand, in D* Lite, the obstacles are denoted in the grid map and cut away possibilities during exploration. Thus D* Lite is not able to utilize a short-cut if the path in the previous knowledge is available during navigation.
Conclusion from the analysis. When utilizing the previous knowledge, BioNav preserves options to utilize possible shortcuts that were not available during exploration in the environment.
However, the D* Lite cuts away options for navigation during exploration, and is not able to utilize possible shortcuts if the previous knowledge from exploration is not obsolete.

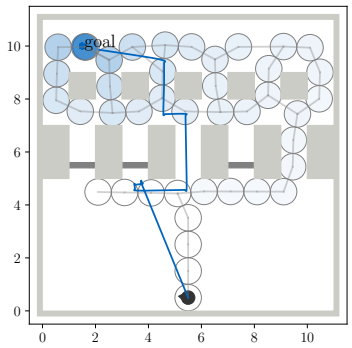
9.5.2 The Time Consumption in Decision Making Is More Concentrated in BioNav than D*Lite

Environment Setup As is shown in figure 9.5. All the gates are open during exploration. Gate 3 and gate 5 are open, other gates are closed during navigation.

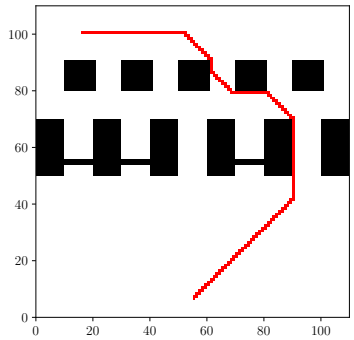
Results of the experiment. The results are shown in figure 9.6 and 9.7. As is shown in the figure, all the agents find the shortcut and go through gate 3 and reach the goal following a



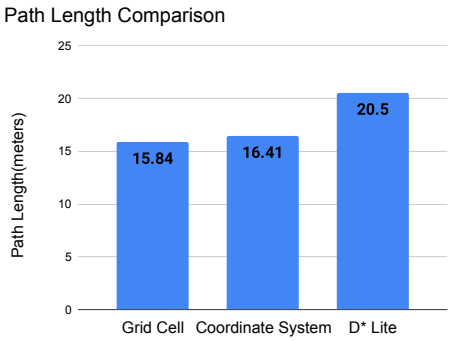
(a) Navigation trajectory of BioNav using grid cell system.



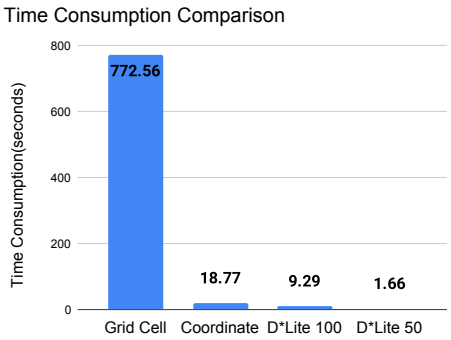
(b) Navigation trajectory of BioNav using coordinate system.



(c) Navigation trajectory of D* Lite.

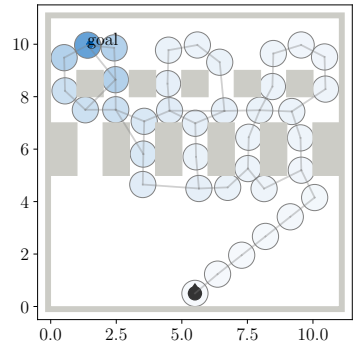


(d) Path length comparison between BioNav and D* Lite.

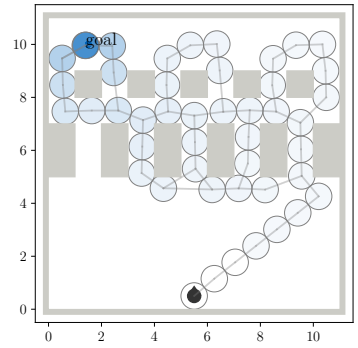


(e) Time consumption comparison between BioNav and D* Lite.

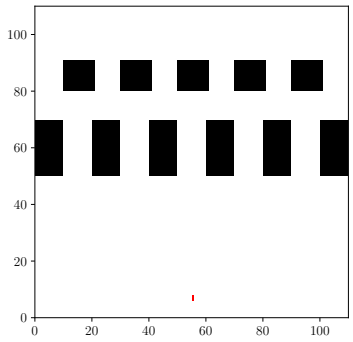
Figure 9.4: This figure shows the navigation trajectory of the agents in the map using the knowledge and the comparison of path length and time consumption.The agent using D* Lite finds the path through gate 5 with very quick decision time, while ignoring the short-cut through gate 2.The agent using BioNav finds the path to the goal through the shortcut through gate 2. The decision time is very long for BioNav using grid cells. The decision time is comparatively much shorter for BioNav using coordinate system. But both versions of the implementation of BioNav travel through similar routes(finds path through gate 2). There are two versions of D* Lite using different precision of obstacle map construction. D*Lite 100 uses map construction of 100×100 cells, while D*Lite 50 uses the map construction of 50×50 cells. The D*Lite using a lower precision is considerably faster than the higher precision one.



(a) Cognitive map of BioNav using grid cell system.

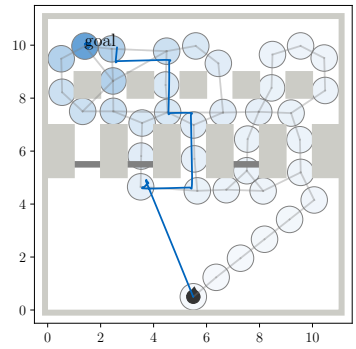


(b) Cognitive map of BioNav using coordinate system.

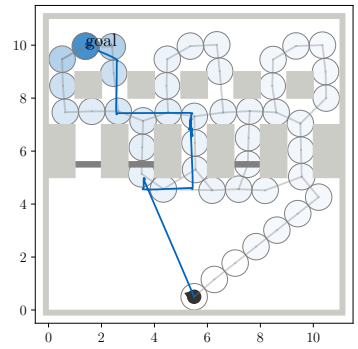


(c) Obstacle map of D* Lite.

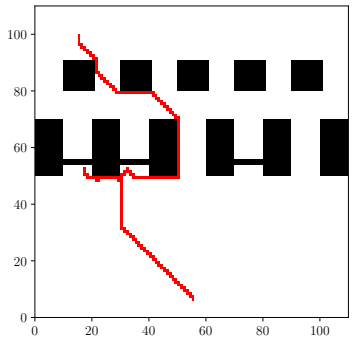
Figure 9.5: This figure shows the knowledge of the agents about the maze.



(a) Navigation trajectory of BioNav using grid cell system

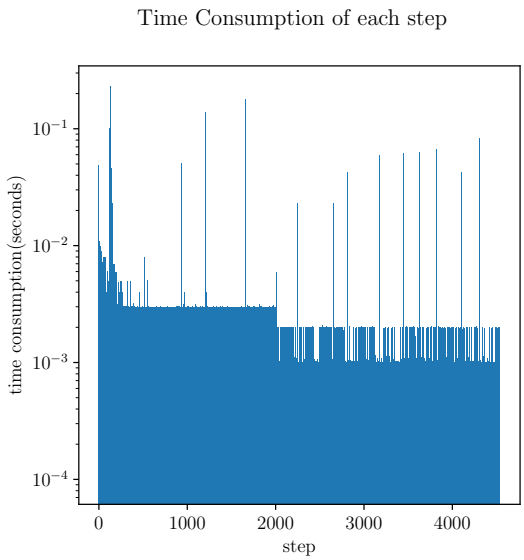


(b) Navigation trajectory of BioNav using coordinate system.

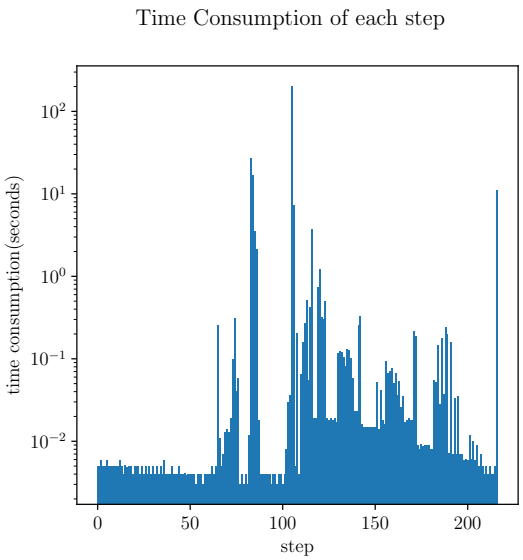


(c) Navigation trajectory of D* Lite.

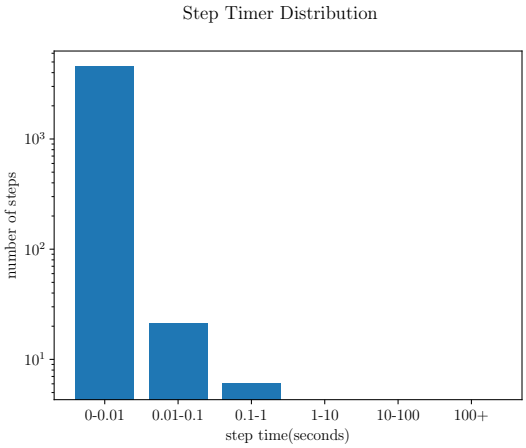
Figure 9.6: This figure shows the navigation trajectory of the agents in the maze using the knowledge. As is shown in the figure, all the agents finds the shortcut and goes through the gate 3 and reaches the goal following similar trajectory.



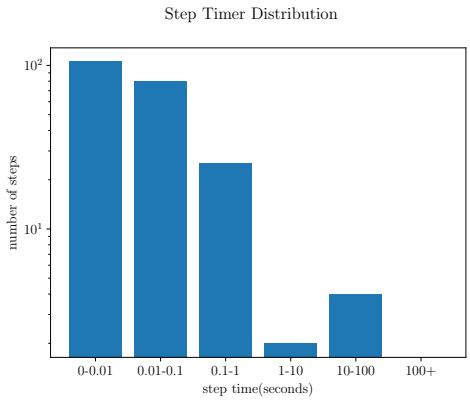
(a) Time consumption of each update step of BioNav using coordinate system



(b) Time consumption of each step of D*Lite 100



(c) Classification of steps according to time consumption of BioNav using coordinate system.



(d) Classification of steps according to time consumption of D* Lite 100.

Figure 9.7: This figure shows the comparison of time consumption statistics by step. a) and b) shows the time consumption of each step of BioNav using coordinate system and D*Lite 100 respectively. c) and d) show the statistics of calculation steps organized by time consumption. As is shown in this figure, BioNav using coordinate system uses much more calculation steps to reach the goal than D* Lite. However, the calculation steps of BioNav is much plainer than D*Lite. If we organize the calculation steps according to its time consumption, it is clear that, even though BioNav uses much more steps than D* Lite, BioNav uses much less time consuming steps. The time consumption of BioNav steps are much more concentrated and has a few time consuming steps during decision making(Linear Look Ahead). The time consumption of D* Lite is much more scattered and makes much more time consuming decisions during navigation.

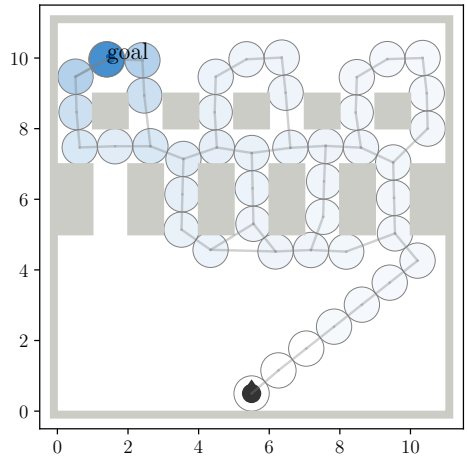
similar trajectory. However, the purpose of this experiment is to analyze the decision time of agent using D* Lite and the agent using BioNav. Because both agents travel through similar trajectories, the comparison is made easy and meaningful. As is shown in figure 9.7, BioNav using coordinate system uses much more calculation steps to reach the goal than D* Lite. However, the calculation steps of BioNav is much plainer than D*Lite. If we organize the calculation steps according to its time consumption, it is clear that, even though BioNav uses much more steps than D* Lite, BioNav uses much less time consuming steps. The time consumption of BioNav steps are much more concentrated and has a few time consuming steps during decision making(Linear Look Ahead). The time consumption of D* Lite is much more scattered and makes much more time consuming decisions during navigation. **Analysis of the results.** This is because D* Lite needs to update its map whenever a new obstacle appears. This implies that D* Lite agents need to perform several map updates, which is very time consuming, for one gate because it discovers the whole gate incrementally. On the other hand, BioNav records the current place cell in the visited cell list and performs a linear look ahead, if the current path is blocked by an obstacle. BioNav performs fewer linear look ahead than D*Lite performs map update. Thus the time consumption of the steps are much more concentrated in BioNav than in D* Lite.
Conclusion from the analysis. D* Lite performs more decision making steps than BioNav. The decision making in BioNav is more concentrated than in D* Lite.

9.5.3 Time Consumption of BioNav Increases Slower than D* Lite When the Problem Becomes More Difficult

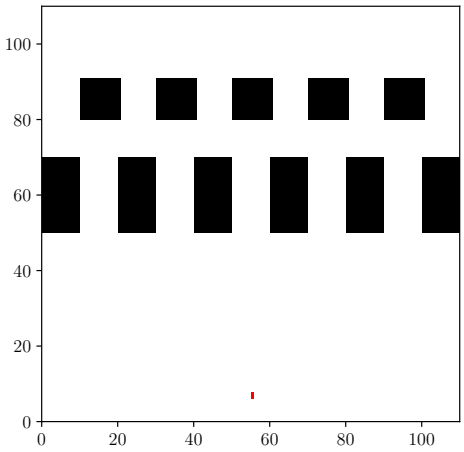
Environment Setup. In this experiment, we gathered the time consumption of agent using BioNav using coordinate system and agent using D* Lite 50 in a different environment setup. The knowledge of the environment is shown in figure 9.8. The following are gathered:

- gate 2 is open, others are closed
- gate 3 is open, others are closed
- gate 4 is open, others are closed
- gate 5 is open, others are closed

Results of the experiment. This experiment aims to analyze the increase in time consumption when the problem becomes more difficult. The results are shown in figure 9.9. As is shown in the figure, the BioNav using coordinate system has a less increase in time consumption when compared to D* Lite, when the problem becomes more and more difficult as the available path deviates further and further away from the shortest path. This corresponds to the statements in chapter 6, where it is analyzed that the D* Lite, which uses A* as path finding algorithms, has a time complexity of $O(\alpha_{env}^4)$, where α_{env}^4 represents the complexity of the problem. On the other hand, BioNav method has a complexity of $O(\alpha_{env})$.
Analysis of the results. As analyzed in the chapter 6, BioNav enjoys an advantage over D* Lite when the complexity of the problem increases. This is because when the depth of the

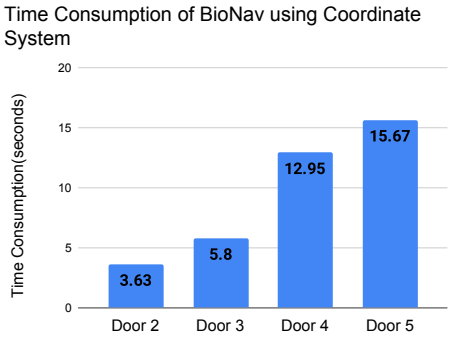


(a) Cognitive map of BioNav using coordinate system.

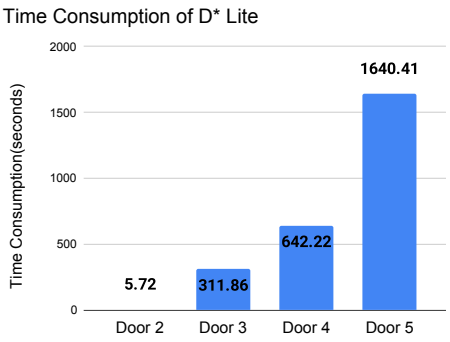


(b) Obstacle map of D* Lite.

Figure 9.8: This figure shows the knowledge of the agents about the maze.



(a) Time consumption of BioNav using coordinate system according to gate setup.



(b) Time consumption of D* Lite 100 according to gate setup.

Figure 9.9: This figure shows the comparison of time consumption between BioNav using coordinate system and D* Lite. As is shown in the figure, the BioNav using coordinate system has a less increase in time consumption when compared to D* Lite, when the problem becomes more and more difficult as the available path deviates further and further away from the shortest path.

solution in to topological graph increases, the nodes that A* path search algorithms need to compute increases at a polynomial speed. However, BioNav only has a linear increase in its decision making. D* Lite relies on keeping an up-to-date obstacle map for obstacle avoidance. D* Lite needs to update all the children nodes of a changed node in its topological graph, when a new obstacle appears. The area of these nodes forms a triangle and has a quadratic increase according to the topological distance of the obstacle node to the goal. However, BioNav uses linear look ahead for obstacle avoidance. The number of times that linear look ahead is performed increases linearly according to the path length, which increases linearly according to the difficulty of the problem. Because the time consumption of a linear look ahead is constant(depending on the pre-set look ahead radius), the time consumption of BioNav only has a linear increase in its time consumption relative to the difficulty of the problem.

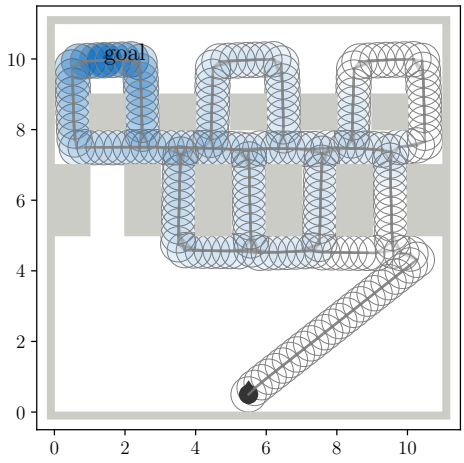
Conclusion from the analysis. The time consumption of BioNav increases slower than D* Lite when the difficulty of the problem increases. The increase of difficulty is reflected in the experiment as larger and larger deviations from the optimal path because of the closed gates.

9.5.4 Time Consumption of BioNav Increases Slower than D* Lite When Precision in Prior Knowledge Increases

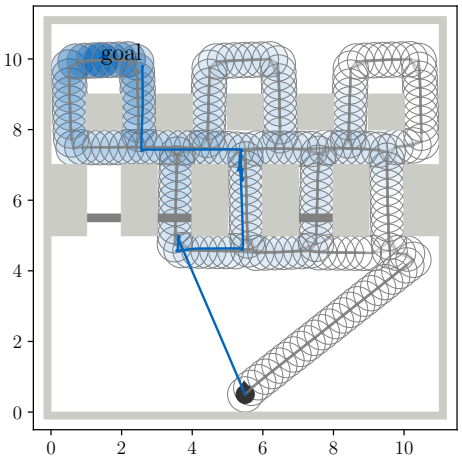
Environment Setup. This experiment aims to show that the BioNav increases slower than D* Lite if the precision of the cognitive map(obstacle map for D* Lite) increases. To increase the precision of the cognitive map of BioNav methods, we need to reduce the range of place cell firing. In BioNav using coordinate system, this is denoted by the *threshold* variable. In the above mentioned experiments, the firing range of the place cell is 1 meter, we reduced it to half of its original value($0.5m$), reconstructed a cognitive map, and compared the navigation time of the method with before. To compare the change of time consumption before and after doubling the precision in D* Lite, we compare the time consumption of D* Lite 50 and D* Lite 100. Setup of the experiment:
All gates are open during exploration.
Gate 3 is open, all other gates are closed during navigation.

Results of the experiment. When the precision is doubled, BioNav method performs better in time consumption than D* Lite. The cognitive map with double precision is shown in figure 9.10. The number of place cells in the cognitive map increases from 46 to 206, corresponding to the 4 times increase in the relative area of the maze. The agent using the cognitive map with double precision is able to navigate to the goal while finding the shortcut through gate 3. As is shown in figure 9.11, the time consumption of BioNav using coordinate system increases two times while the time consumption of D* Lite increases eight times.

Conclusion and analysis. When the precision increase, the time consumption of BioNav increases more slowly than D* Lite. This is because doubling the precision is comparative to doubling the side length of the maze. The topological depth of the goal is also doubled. As is stated in section 9.5.3, the BioNav has a linear increase of time consumption relative to the topological depth of the goal, while D* Lite has a polynomial increase in time consumption. However, in addition to the increase in topological depth of the goal, the number of place

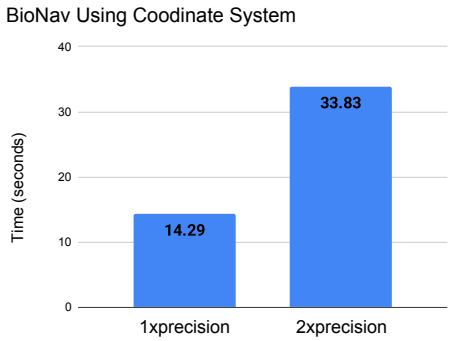


(a) The cognitive map of the maze with double precision.

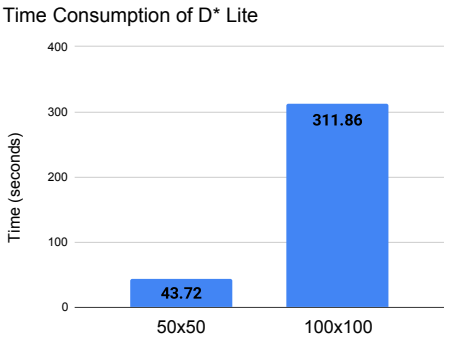


(b) The navigation trajectory of the agent to the goal.

Figure 9.10: This figure demonstrates the increase in precision of the cognitive map. As is shown in the figure, the place cells represent smaller areas and become much denser. The cognitive map with single precision has 46 place cells while the cognitive map with double precision has 206 place cells. The agent uses the cognitive map of double precision to navigate to the goal while finding the shortcut through gate 3. The navigation trajectory is marked by the blue line in figure b).

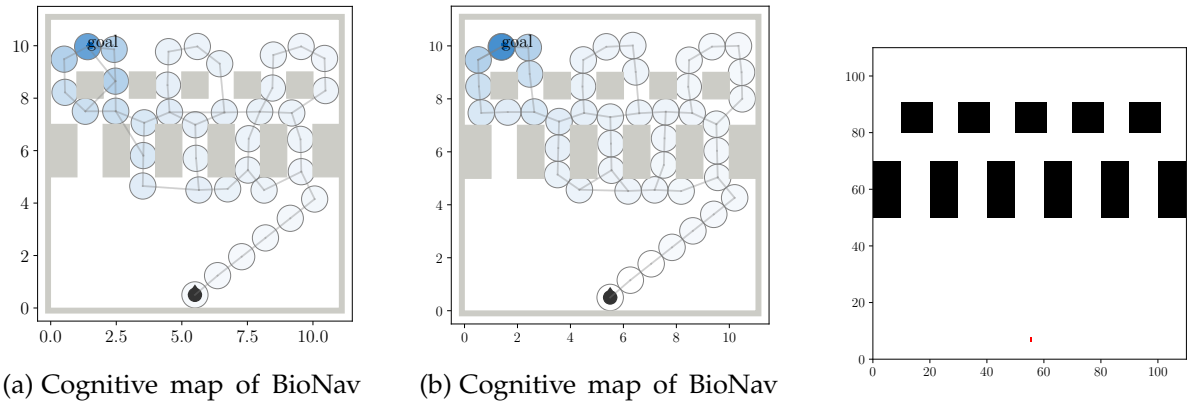


(a) Comparison of time consumption of BioNav using coordinate system.



(b) Comparison of time consumption of D* Lite.

Figure 9.11: This figure demonstrates the increase in time consumption of BioNav using coordinate system and D* Lite, after doubling the precision of knowledge. In figure a), the time consumption of BioNav increases two times after doubling the precision while the time consumption of D* Lite increases eight times.



(a) Cognitive map of BioNav using grid cell system.

(b) Cognitive map of BioNav using coordinate system.

(c) Obstacle map of D* Lite.

Figure 9.12: This figure shows the knowledge of the agents about the maze.

cells also increases significantly. The increased number of place cells makes each update step and linear look ahead more time consuming. This explains why the time consumption increases more dramatically relative to precision than difficulty of the problem.

9.6 Similarities between BioNav and D* Lite

9.6.1 Both BioNav and D* Lite Performs Well When the Actual Environment Matches the Knowledge

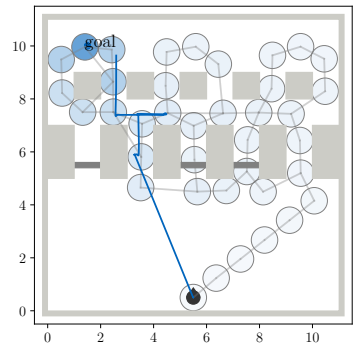
Environment Setup. As is shown in figure 9.12, all the gates are open during exploration. Gate 2 and 5 open, other gates are closed during navigation.

Results of the experiment. The results are shown in figure 9.13. The agent using D* Lite finds the path through gate 2 with very quick decision time.

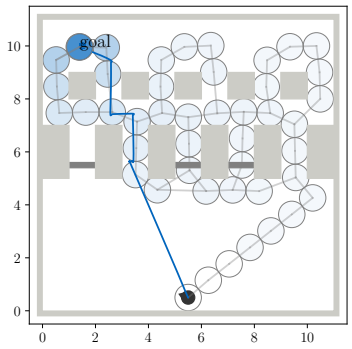
The agent using BioNav also finds the path to the goal through gate 2. The decision time is very long for BioNav using grid cells. The decision time is very quick for BioNav using coordinate system. But both version of implementation of BioNav travel through similar routes(finds path through gate 2).

Analysis of the results. D* Lite runs very fast when the default route in its knowledge graph(the shortest path according to the 2D obstacle grid map during exploration) is available during navigation. The reason behind this is that, D* Lite does not need to perform updates to its 2D grid. The changes in the environment during navigation is either not observed by the agent, or it is irrelevant and thus ignored by the D* Lite during the focus(an irrelevant change is changes to the edges that have a larger key value than the current key, and stored in a lower position in the heap. Thus, they are not updated.)

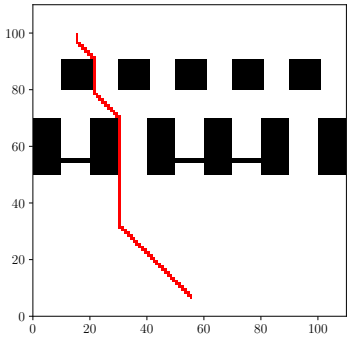
The BioNav performs very well in this problem, because this problem is a simple problem. The setup of the doors corresponds to the optimistic assumption of BioNav(BioNav assumes a minimum number of obstacles in the environment). Thus, very small number of obstacle avoidance need to be performed to solve this problem.



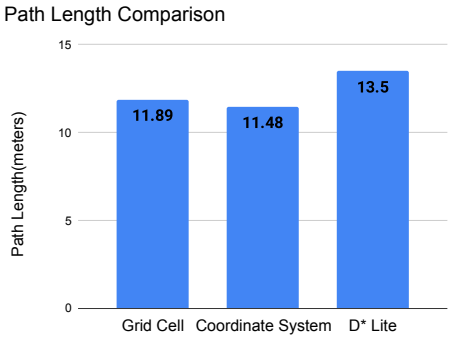
(a) Navigation trajectory of BioNav using grid cell system.



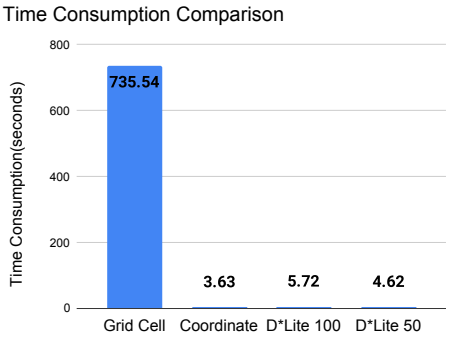
(b) Navigation trajectory of BioNav using coordinate system.



(c) Navigation trajectory of D* Lite.



(d) Path length comparison between BioNav and D* Lite.



(e) Time consumption comparison between BioNav and D* Lite.

Figure 9.13: This figure shows the navigation trajectory of the agents in the map using the knowledge, and also the comparison of path length and time consumption. The agent using D* Lite finds the path through gate 2 with very quick decision time. The agent using BioNav also finds the path to the goal through gate 2. The decision time is very long for BioNav using grid cells. The decision time is very quick for BioNav using coordinate system. But both versions of the implementation of BioNav travel through similar routes(finds path through gate 2).

Conclusion from the analysis. D* Lite runs fast and selects the optimal pathway when the environment setup and the optimal route matches the optimal route in its exploration. BioNav runs fast when the environment setup matches its optimistic assumption of minimum obstacles.

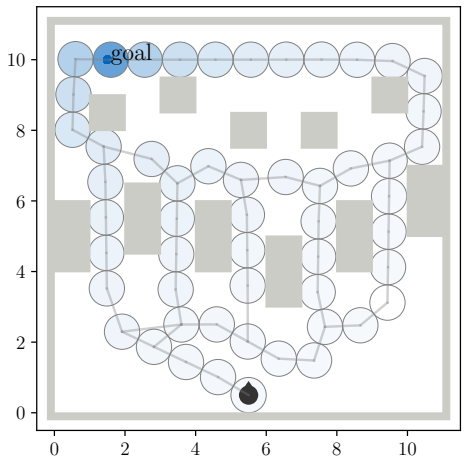
9.7 Weakness of BioNav Versus D* Lite

9.7.1 D* Lite performs Better than BioNav in Unstructured Environments

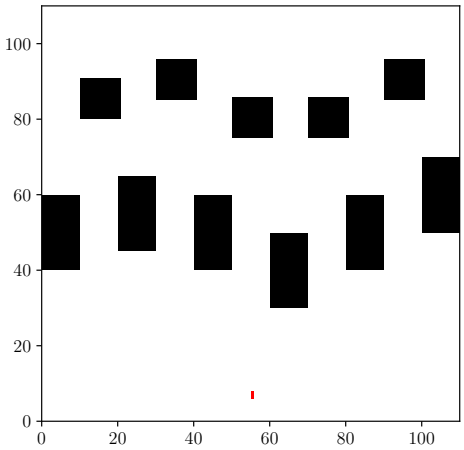
Environment Setup This experiment aims to test the ability of the agent using BioNav and D* Lite to navigate in an unstructured environment. The pathways and columns, which were aligned in the same line, now have an offset to its original position. The setup of the gates in the pathway can still be changed. In this experiment, the agent builds up knowledge by exploring the unstructured maze when all gates are open. Then the agent tries to navigate through the unstructured maze, where only gate 5 is open, to reach its goal. **Results of the experiment.** As shown in figure 9.14, the agent using BioNav was not able to navigate to its goal while the agent using D* Lite is able to navigate to its goal.
Analysis of the results. BioNav methods are not able to reach the goal because the plane was split into two halves by an obstacle. During linear look ahead, the agent does not consider obstacles and keeps aiming at the places cells located in the other half of the maze. The agent keeps running into the obstacle splitting the maze because it can not suppress the place cells that are attracting it to that direction by visiting them.
Conclusion from the analysis. BioNav can not perform navigation in unstructured maze, while D* Lite can.

9.8 Summary of the Comparison Experiments

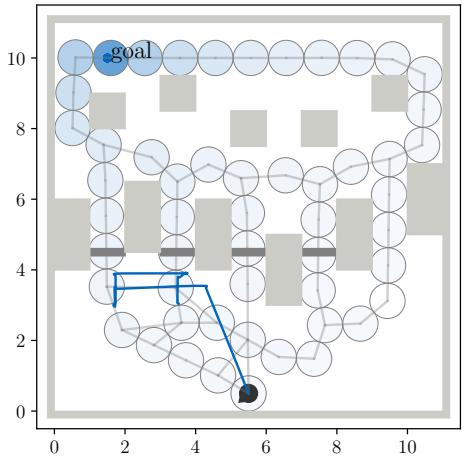
Through comparison experiments, we are able to conclude that BioNav and D* Lite has their own strengths and weaknesses. In the domain of functionality, BioNav shows stronger ability in shortcut finding while D* Lite is more adaptive to unstructured environments. In the domain of performance in time consumption, the BioNav, especially BioNav using coordinate system, shows better performance than D* Lite. The time consumption of BioNav increases more slowly relative to increase in both difficulty of the problem and precision. When the environment settings match the expectations of both methods, both methods perform very well. The summary of the conclusions from the comparison experiments are listed in the table below.



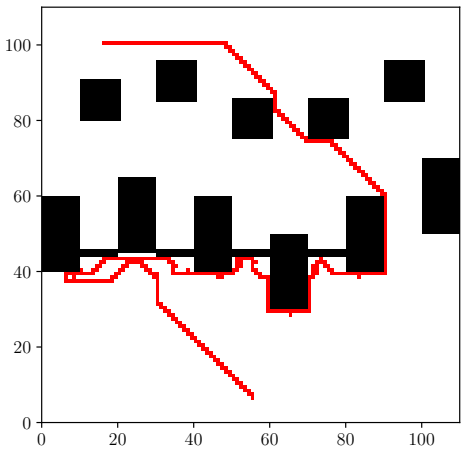
(a) The cognitive map of BioNav using coordinate system.



(b) The obstacle map of D* Lite



(c) The navigation trajectory of BioNav using coordinate system.



(d) The navigation trajectory of D* Lite.

Figure 9.14: This figure shows the prior knowledge and navigation trajectory of BioNav using coordinate system and D* Lite in an unstructured environment. The agent using BioNav was not able to navigate to its goal while the agent using D* Lite is able to navigate to its goal.

Navigation Methods		
Aspects of Comparison	BioNav	D* Lite
Shortcut Discovery	Stronger	Weaker
Time Consumption Relative to Difficulty	Linear	Polynomial
Time Consumption Relative to Precision	Moderate	Huge
Environment Matching Expectation	Fast	Fast
Unstructured Environment	Badly	Well

10 Conclusion and Future Work

In this chapter, we will summarize our work. We will also give an outlook on the possible interesting future works that are derived from the findings in this paper.

10.1 Conclusion

In this research, we aimed to fix the problems in the baseline method and make comparison experiments to prove the strength of our method against conventional methods. With our work, we have successfully accomplished our goal.

After analyzing the baseline method, there are three main problems:

- The baseline method lacks flexibility in a dynamic environment.
- The baseline method lacks accuracy in its navigation methods.
- The baseline method demands too much computation power and time consumption.

We have concluded reasonable causes of these problems through our analysis and accomplished valid solutions to these problems. The baseline method lacks flexibility in a dynamic environment, because it is not able to utilize the information about visited places. We added the visited cell list to the system, which prevents the system from being trapped in visited places. The baseline method lacks accuracy in its navigation methods, because the grid cell system, which is responsible for path integration, lacks diversity in its spiking characteristics of grid cell modules. We studied the factors affecting the diversities of grid cell modules and successfully arranged the grid cell system with more diverse spiking characteristics using different g_m values and heading alignments. We also proposed a new method for the weight distribution among grid cell modules. These improvements proved to be effective and appropriate. The baseline method consumes too much computation power and time, because the baseline method uses a very complex system, namely grid cell system, for path integration. We analyzed the time consumption of the baseline method using a simple mathematical model and implemented two methods to reduce time consumption. The first method makes the update of the grid cell system parallel by implementing a GPU version of the grid cell system. Even though significant reduction in time consumption is achieved in the single model of grid cell system, the reduction becomes less significant after integrating the parallel grid cell system to other systems. This is because there are other bottlenecks and overhead in the system, namely the interface between the place cell system and the grid cell system. In the second method, we used a simpler system, namely the coordinate system, to replace the time consuming grid cell system. This method has reduced the time consumption very

significantly. This reduction in time consumption remains significant after integrating it to other systems. From the second method, we derived the second version of our system: BioNav using a coordinate system.

After fixing the problems in the baseline methods, we have conducted comparison experiments between BioNav and D* Lite. D* Lite is a good representative of well established conventional methods, because it has similar characteristics with BioNav and is adaptable to similar experiment setups. We converted the testing environment of BioNav to make it suitable for experiments on D* Lite. Through multiple experiments, we have demonstrated the strength of BioNav system against D* Lite through convincing results. We were able to show that BioNav have strength against D* Lite in not only functionalities but also performance. We have demonstrated that the BioNav has stronger shortcut finding ability and better performance when both the difficulty of the problem and the precision of the environment increases. However, the difficulty of BioNav in navigating in an unstructured environment is also demonstrated in the experiment.

Our work helps understand the benefits and problems in integrating biologically inspired methods into robotics. Our work has also pointed out possible solutions for problems during the integration.

10.2 Future Work

We want to highlight several interesting directions for future research and improvements.

Spiking characteristics of grid cell modules. Even though we have proposed methods of adding diversities to the grid cell system and factors affecting the p=spiking characteristics of grid cell modules. However, it could be interesting how exactly these factors interact with each other to produce different spiking characteristics.

Parallel place cell system. We have achieved a parallel grid cell system using GPU programming. However, as is shown in the paper, other bottlenecks, like the place cell system, still exist in the system. Reducing the time consumption in these bottlenecks could yield significant reductions in time consumption.

Unstructured environments. As is shown in the experiment, BioNav can not adapt in unstructured environments very well. It is meaningful to come up with new improvements to make the system adaptable to less structured environments.

Better combination of navigation methods. BioNav currently uses two navigation methods: vector based and topology based. It could be meaningful to come up with better ways to combine and utilize the instructions from both methods.

List of Figures

Bibliography

[1] J. O'Keefe and J. Dostrovsky. "The hippocampus as a spatial map. Preliminary evidence from unit activity in the freely-moving rat". In: *Brain Research* 34.1 (Nov. 1971), pp. 171–175. DOI: 10.1016/0006-8993(71)90358-1. URL: https://doi.org/10.1016/0006-8993(71)90358-1.

[2] J. S. Taube. "The Head Direction Signal: Origins and Sensory-Motor Integration". In: *Annual Review of Neuroscience* 30.1 (July 2007), pp. 181–207. DOI: 10.1146/annurev.neuro.29.051605.112854. URL: https://doi.org/10.1146/annurev.neuro.29.051605.112854.

[3] T. Hafting, M. Fyhn, S. Molden, M.-B. Moser, and E. I. Moser. "Microstructure of a spatial map in the entorhinal cortex". In: *Nature* 436.7052 (June 2005), pp. 801–806. DOI: 10.1038/nature03721. URL: https://doi.org/10.1038/nature03721.

[4] N. Burgess, A. Jackson, T. Hartley, and J. O'Keefe. "Predictions derived from modelling the hippocampal role in navigation". In: *Biological Cybernetics* 83.3 (Aug. 2000), pp. 301–312. DOI: 10.1007/s004220000172. URL: https://doi.org/10.1007/s004220000172.

[5] Ø. A. Høydal, E. R. Skytøen, S. O. Andersson, M.-B. Moser, and E. I. Moser. "Object-vector coding in the medial entorhinal cortex". In: *Nature* 568.7752 (Apr. 2019), pp. 400–404. DOI: 10.1038/s41586-019-1077-7. URL: https://doi.org/10.1038/s41586-019-1077-7.

[6] J. C. Whittington, T. H. Muller, S. Mark, G. Chen, C. Barry, N. Burgess, and T. E. Behrens. "The Tolman-Eichenbaum Machine: Unifying Space and Relational Memory through Generalization in the Hippocampal Formation". In: *Cell* 183.5 (Nov. 2020), 1249–1263.e23. DOI: 10.1016/j.cell.2020.10.024. URL: https://doi.org/10.1016/j.cell.2020.10.024.

[7] J. C. R. Whittington, D. McCaffary, J. J. W. Bakermans, and T. E. J. Behrens. "How to build a cognitive map". In: *Nature Neuroscience* 25.10 (Sept. 2022), pp. 1257–1272. DOI: 10.1038/s41593-022-01153-y. URL: https://doi.org/10.1038/s41593-022-01153-y.

[8] C. Sun, W. Yang, J. Martin, and S. Tonegawa. "Hippocampal neurons represent events as transferable units of experience". In: *Nature Neuroscience* 23.5 (Apr. 2020), pp. 651–663. DOI: 10.1038/s41593-020-0614-x. URL: https://doi.org/10.1038/s41593-020-0614-x.

[9] V. E. and. "A Passive Mechanism for Goal-Directed Navigation using Grid Cells". In: *07/20/2015-07/24/2015*. The MIT Press, July 2015. DOI: 10.7551/978-0-262-33027-5-ch039. URL: https://doi.org/10.7551/978-0-262-33027-5-ch039.

[10] V. Edvardsen, A. Bicanski, and N. Burgess. "Navigating with grid and place cells in cluttered environments". In: *Hippocampus* 30.3 (Aug. 2019), pp. 220–232. DOI: 10.1002/hipo.23147. URL: https://doi.org/10.1002/hipo.23147.

[11] R. Ivey, D. Bullock, and S. Grossberg. "A neuromorphic model of spatial lookahead planning". In: *Neural Networks* 24.3 (Apr. 2011), pp. 257–266. DOI: 10.1016/j.neunet.2010.11.002. URL: https://doi.org/10.1016/j.neunet.2010.11.002.

[12] T. Engelmann. "Biologically inspired spatial navigation using vector-based and topology-based path planning". In: (2022).

[13] F. Sargolini, M. Fyhn, T. Hafting, B. L. McNaughton, M. P. Witter, M.-B. Moser, and E. I. Moser. "Conjunctive Representation of Position, Direction, and Velocity in Entorhinal Cortex". In: *Science* 312.5774 (May 2006), pp. 758–762. DOI: 10.1126/science.1125572. URL: https://doi.org/10.1126/science.1125572.

[14] E. C. Tolman. "Cognitive maps in rats and men." In: *Psychological Review* 55.4 (1948), pp. 189–208. DOI: 10.1037/h0061626. URL: https://doi.org/10.1037/h0061626.

[15] E. I. Moser, E. Kropff, and M.-B. Moser. "Place Cells, Grid Cells, and the Brain's Spatial Representation System". In: *Annual Review of Neuroscience* 31.1 (July 2008), pp. 69–89. DOI: 10.1146/annurev.neuro.31.061307.090723. URL: https://doi.org/10.1146/annurev.neuro.31.061307.090723.

[16] E. I. Moser, M.-B. Moser, and B. L. McNaughton. "Spatial representation in the hippocampal formation: a history". In: *Nature Neuroscience* 20.11 (Nov. 2017), pp. 1448–1464. DOI: 10.1038/nn.4653. URL: https://doi.org/10.1038/nn.4653.

[17] H. Eichenbaum. "Memory: Organization and Control". In: *Annual Review of Psychology* 68.1 (Jan. 2017), pp. 19–45. DOI: 10.1146/annurev-psych-010416-044131. URL: https://doi.org/10.1146/annurev-psych-010416-044131.

[18] U. M. Erdem and M. Hasselmo. "A goal-directed spatial navigation model using forward trajectory planning based on grid cells". In: *European Journal of Neuroscience* 35.6 (Mar. 2012), pp. 916–931. DOI: 10.1111/j.1460-9568.2012.08015.x. URL: https://doi.org/10.1111/j.1460-9568.2012.08015.x.

[19] R. Place, A. Farovik, M. Brockmann, and H. Eichenbaum. "Bidirectional prefrontal-hippocampal interactions support context-guided memory". In: *Nature Neuroscience* 19.8 (Aug. 2016), pp. 992–994. DOI: 10.1038/nn.4327. URL: https://doi.org/10.1038/nn.4327.

[20] B. G. Burton, V. Hok, E. Save, and B. Poucet. "Lesion of the ventral and intermediate hippocampus abolishes anticipatory activity in the medial prefrontal cortex of the rat". In: *Behavioural Brain Research* 199.2 (May 2009), pp. 222–234. DOI: 10.1016/j.bbr.2008.11.045. URL: https://doi.org/10.1016/j.bbr.2008.11.045.

[21] V. Hok, E. Save, P. P. Lenck-Santini, and B. Poucet. "Coding for spatial goals in the prelimbic/infralimbic area of the rat frontal cortex". In: *Proceedings of the National Academy of Sciences* 102.12 (Mar. 2005), pp. 4602–4607. DOI: 10.1073/pnas.0407332102. URL: https://doi.org/10.1073/pnas.0407332102.

[22] S. Koenig and M. Likhachev. "Fast replanning for navigation in unknown terrain". In: *IEEE Transactions on Robotics* 21.3 (2005), pp. 354–363.

[23] V. Edvardsen. "Long-range navigation by path integration and decoding of grid cells in a neural network". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE, May 2017. DOI: 10.1109/ijcnn.2017.7966406. URL: https://doi.org/10.1109/ijcnn.2017.7966406.

[24] M. Deyo. *Mdeyo/D-star-lite: Python implementation of D* lite with interactive visualizer for grid path-planning example*. Mar. 2018. URL: https://github.com/mdeyo/d-star-lite.