AUTONOMOUS NAVIGATION USING REINFORCEMENT LEARNING WITH SPIKING NEURAL NETWORKS

A Thesis

by

AMARNATH MAHADEVUNI

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Chair of Committee,	Peng Li
Co-Chair of Committee,	Paul Gratz
Committee Members,	Alireza Talebpour
Head of Department,	Miroslav Begovic

May 2018

Major Subject: Computer Engineering

ABSTRACT

The autonomous navigation of mobile robots is of great interest in mobile robotics. Algorithms such as simultaneous localization and mapping (SLAM) and artificial potential field methods can be applied to known and mapped environments. However, navigating in an unknown, and unmapped environments is still a challenge. In this research, we propose an algorithm for mobile robot navigation in the near-shortest possible time toward a predefined target location in an unknown environment containing obstacles. The algorithm is based on a reinforcement learning paradigm with biologically realistic spiking neural networks. We make use of eligibility traces that are inherent to spiking neural networks to solve the delayed reward problem implicitly present in reinforcement learning. With this algorithm, we achieve a set of movement decisions for the mobile robot to reach the target in the near-shortest time.

DEDICATION

To my parents, my friends, and my professors.

ACKNOWLEDGMENTS

I would like to express my gratitude to my thesis advisor Dr. Peng Li for his continuous guidance through out the research. I would like to thank my thesis co-chair Dr. Paul Gratz and committee member Dr. Alireza Talebpour for being on my committee and providing constructive feedback for my work. I am also grateful to my friends Paul Crother, Seungjai Ahn, and Myung Seok Shim in the research group for their work in the area, and the help I have received from them. I express my appreciation for the faculty members of Texas A&M University for providing me excellent knowledge and experience in various areas related to my research. I would like to also thank my family and friends for encouraging me in this pursuit.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

This work was supported by a thesis committee consisting of Professor Dr. Peng Li [advisor] and Dr. Paul Gratz of the Department of Electrical and Computer Engineering and Professor Dr. Alireza Talebpour of the Department of Civil Engineering.

I have received continuous feedback from Dr. Peng Li which helped me improve my work throughout the research. Dr. Paul Gratz and Dr. Alireza Talebpour have provided valuable suggestions to analyze the certain aspects of the research.

All the work conducted for the thesis was completed by the student independently.

Funding Sources

This work was supported by the National Science Foundation under Grant No. CCF-1639995 and the Semiconductor Research Corporation (SRC) under Task 2692.001.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

1.1 Background and Motivation

Autonomous navigation is a technology which enables vehicles to move on their own to achieve their navigation goals without human intervention. Technological developments in autonomous navigation are applicable in the following areas :

1. **Self-driving cars :** A self-driving car can drive itself by utilizing a set of sensors, actuators, intelligent algorithms, and great amount of data from its own experiences of driving or from the driving experience of other cars or individuals [2]. It also utilizes maps to plan a path from a start location to destination. The potential advantages of self-driving cars include :

   (a) Road safety : Accidents happen due to human errors such as large reaction time, disregard for traffic rules, insufficient driving skills to mention a few. A self driving car can eliminate the risk of accidents by being fast and predictable. The coordination between multiple self driving cars can happen faster than between human drivers and can lead to collision free navigation.

   (b) Saves effort : With self driving cars, one can better utilize the time spent in driving for some other productive purpose.

   (c) Saves money : Self driving technology can eliminate the need for skilled human drivers, thereby saving money that would otherwise be spent on driver salaries.

2. **Extraterrestrial exploration :** An exciting use of autonomous navigation is in areas where it is difficult for humans to explore. Extraterrestrial exploration is one such domain where humans cannot easily conduct research directly. This includes

space exploration as well as planetary exploration. The research conducted could be for traces of water, life, any resource useful to humanity. As such intelligent mobile systems are needed to explore the unknown terrain of planetary bodies. Navigation is an important part in exploration because it helps collect geospatial information regarding resources of interest. The use of intelligent unmanned vehicles can greatly reduce human effort in such endeavors. As such research to build autonomous navigation systems for extraterrestrial exploration is of great interest [3] [4].

3. **Household robotics :** Autonomous systems can save time and effort even at household level. One can imagine building a robot that cooks food, cleans the floor, or conduct household repairs for example. In particular, the house cleaning robots require autonomous navigation to perform a thorough job. One such device is "Roomba" from "iRobot" [5]. This device is a mobile vacuum cleaner that moves around in a house and vacuum cleans the floor. We can imagine building mobile humanoid robot assistants performing several logistic tasks.

4. **Search and Rescue missions :**In situations such as earthquakes, landslides, floods, or any other such calamity, response time in search and rescue missions is crucial to save victims. Machines are inherently more durable and faster than humans when it comes to mobility. Therefore building robotic systems for search and rescue missions is a promising research area. In certain situations where it is difficult or impossible for human personnel to conduct a search, we can rely on intelligent robotic systems designed to quickly explore the area, identify the victims, and relay the information to the rescue personnel [6] . So autonomous navigation is applicable to robots in search and rescue mission in making them move faster and conduct an efficient search.

1.1.1 Machine Learning for Autonomous Navigation

Navigation is a complex task. As such, an autonomous vehicle requires intelligent algorithms to detect objects in an image and sophisticated control mechanisms embedded into its software and hardware.

In the recent years, with increase in computation power, machine learning has proven to be a promising technique in handling complex tasks relevant to navigation such as object recognition in images, and mechanical control. In self driving cars, object recognition can be mainly used to detect vehicles, pedestrians, traffic signals and road signs. Mechanical control is essential to drive in a smooth fashion without sudden jerks. Supervised machine learning techniques are used to learn to detect different types of objects in an image [7]. Control mechanisms are learned using reinforcement learning [8] [9], another form of machine learning. Reinforcement learning is the backbone for the research described in this document. Chapter 2 discusses reinforcement learning in greater detail.

1.1.2 Motivation and Research Goals

In general, complex tasks are handled by breaking them down into smaller problems. Further, one can define simplified versions of each of these smaller problems by keeping the essential aspects of it and abstracting away irrelevant details. The autonomous navigation problem has many different aspects to it such as object detection, environment mapping, path planning, navigation time, obstacle sensing to name a few. In this research, a particular part of navigation, "navigation time" is considered. It is essential for autonomous vehicles to reach their destination in a shortest amount of time possible. The reasons for this could be different for different application areas:

1. **Self driving cars :** It is essential for self driving cars to get to a destination in short time to save fuel as well as passengers time. The vehicle will have to use its knowledge of the environment from its past experience as well as the current

situation to make better navigation choices leading to shorter navigation time than normal human driving.

2. **Extra terrestrial exploration :** In this application, a shorter navigation time to a site could mean faster exploration results.

3. **Household robotics :** In devices such as the vacuum cleaner robot previously mentioned, battery life is important, which determines how much of the house the robot can clean before it gets back to the charging dock. Therefore shorter navigation times to desired locations on the floor can save battery time and the robot can do more cleaning than with longer navigation times.

4. **Search and rescue missions :** The survival of an individual affected by a calamity depends on how early they are discovered and rescued. When search and rescue robots are deployed, it is desired that they arrive at the disaster location as quickly as possible so they can discover and inform the rescue personnel faster, thereby saving lives.

Therefore in applications of autonomous navigation, the aspect of navigation time provides a promising area of research. Thus this thesis addresses the issue of finding the shortest navigation time path for the navigation task in a simplified form as described below :

1. The navigation environment under consideration is a 2 dimensional world.

2. The navigating device is a differential drive mobile robot equipped with proximity sensors.

3. Rectangular obstacles placed randomly in the environment.

4. A destination location referred to as the "Target" or the "Goal" location.

5. The robot navigates autonomously from a "Start" location to the "Goal" location.

6. The navigation time is then the time taken for the robot to reach the "Goal" location from a given "Start" location.

The description is illustrated in Figure 1.1

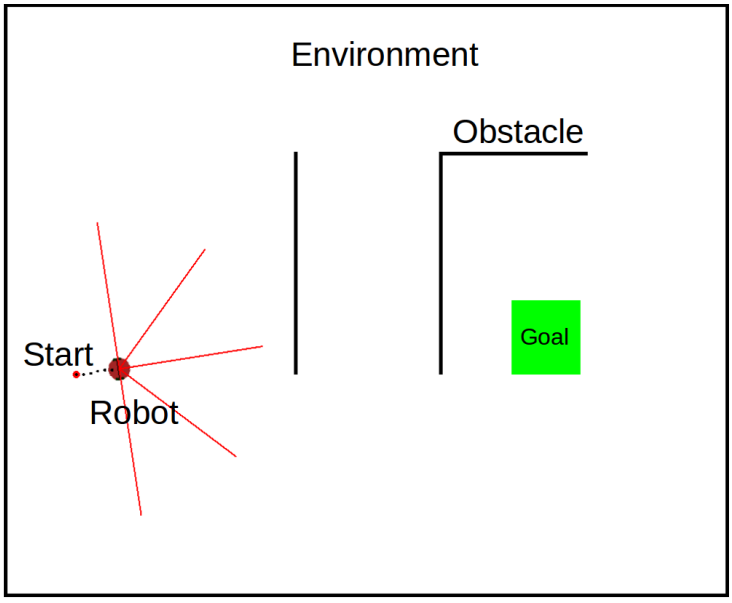

Figure 1.1: Example instance of the problem. Reprinted with permission from [1]. ©[2017] IEEE.

The challenges faced in achieving this goal are :

1. Environment is unknown to the robot.

2. The size and location of obstacles are unknown to the robot.

These challenges call for a trial and error approach to solve the problem. Currently the best way to solve a problem using trial and error approach is to formulate the problem as a

reinforcement learning problem [8],and find ways and means to implement the solution. In chapter 5, we shall discuss how animals exhibit reinforcement learning naturally. They do so using biologically realistic neurons called the spiking neurons that make up an animal brain. Spiking neurons are discussed in detail in chapter 4. This research is taken as an opportunity to explore biologically realistic mechanisms to implement reinforcement learning. Therefore we propose a biologically inspired algorithm using spiking neural networks to implement the reinforcement learning solution.

Thus in order to achieve the goal of finding the shortest possible navigation time path for a scenario similar to Figure 1.1, the following research goals are proposed.

1. Understanding the physics of differential drive mobile robot navigation (Chapter 2).
2. Researching the reinforcement learning framework (Chapter 3).
3. Explore spiking neurons and networks of spiking neurons (Chapter 4).
4. Understanding how reinforcement learning happens in spiking neural networks (Chapter 5).
5. Discuss existing solutions to the navigation problem. (Chapter 6).
6. Formulate the problem of minimizing the navigation time as a reinforcement learning problem (Chapter 7).
7. Propose a complete solution combining robot dynamics with reinforcement learning implemented using spiking neural networks to solve the problem (Chapter 8).
8. Discuss experimental setup and simulation results (Chapter 9).

Chapters 2,3, and 4 are independent of one another. Chapter 5 requires material in chapters 3 and 4. Chapter 6 requires material in chapter 2. Chapter 7, 8, and 9 must be read last in that order.

2. DYNAMICS OF DIFFERENTIAL DRIVE MOBILE ROBOT

The research experiments are conducted in a simulated environment. The model of the mobile robot used in the simulations is a differential drive model [10]. It is a simplistic model in which the robot body has two wheels that can have different angular speeds. The structure and parameters of differential drive robot are discussed next.

2.1 Differential Drive Abstraction

The abstraction for the differential drive robot is shown in Figure 2.1

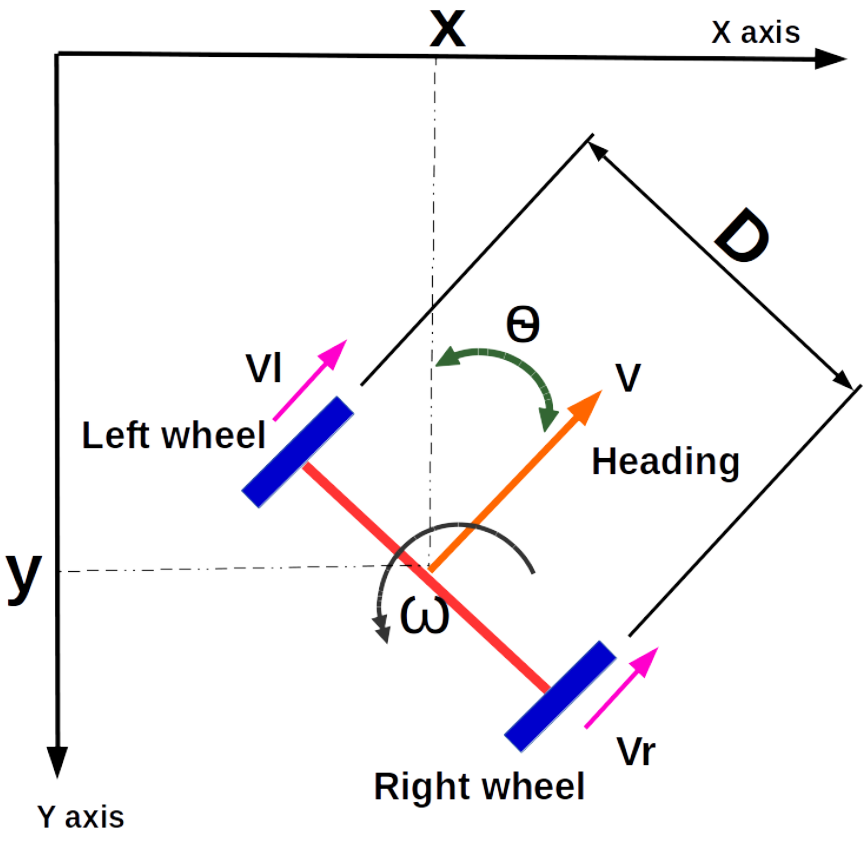

Figure 2.1: Differential drive abstraction.

It is described by the following parameters :

- A cartesian coordinate system with X-Y axes.
- (x, y) coordinates for the center of the robot.
- V_l : linear speed of left wheel.
- V_r : linear speed of right wheel.
- V : linear speed of the robot's center.
- θ : angular position.
- ω : angular speed.
- D : separation between the wheels.

In order to display the robot in a graphics based simulation environment, an attribute pose is defined as follows :

$$Pose = (x, y, \theta) \tag{2.1}$$

If the pose of the robot is known, its position and orientation can be set in a graphics environment.

2.2 Dynamics

A physical differential drive robot is controlled using the parameters V_l and V_r. They are related to its angular speed as follows :

$$\omega = \frac{V_r - V_l}{D} \tag{2.2}$$

Therefore the change in angular position with this angular speed after duration δt is given by

$$\delta\theta = \omega\delta t \tag{2.3}$$

Also for linear speed of the robot's center

$$V = \frac{V_r + V_l}{2} \tag{2.4}$$

It is important to know how the differential drive robot can change its pose with this model. Figure 2.2 shows the robot at Pose1 and Pose2.

The differential drive robot uses the three step approach [10] to change its pose from Pose1 to Pose2 :

1. Rotate the robot inplace to point to its new pose location (x_2, y_2) using :

$$V_l = -V_r \tag{2.5}$$

$$\implies \omega = \frac{2V_r}{D} \tag{2.6}$$

$$\implies \delta\theta = \frac{2V_r}{D}\delta t \tag{2.7}$$

$$\implies V_r = \frac{D\delta\theta}{2\delta t} \tag{2.8}$$

   Point the robot to its new location using $V_r = -V_l$ with V_r calculated as shown in equation 2.8

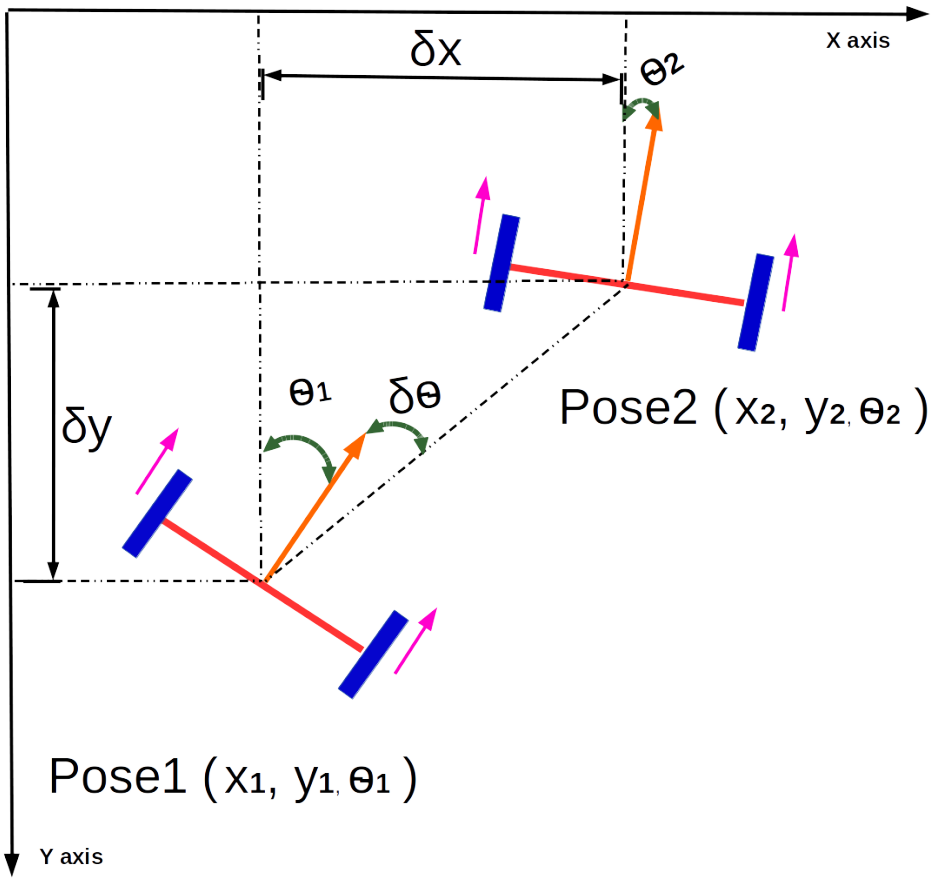

Figure 2.2: Changing the pose of a differential drive robot.

2. Linearly translate the robot to its new pose location (x_2, y_2) using :

$$V_l = V_r \tag{2.9}$$

$$\implies V = V_r \tag{2.10}$$

$$\implies \delta x = V_r cos(\theta + \delta\theta)\delta t \tag{2.11}$$

$$\implies \delta y = -V_r sin(\theta + \delta\theta)\delta t \tag{2.12}$$

$$\implies V_r = \frac{\sqrt{\delta x^2 + \delta y^2}}{\delta t} \tag{2.13}$$

Therefore translate the robot to its new location using $V_r = V_l$ with V_r calculated as shown in equation 2.13

3. Rotate the robot inplace so that the new angular position is θ_2 on the lines of equations 2.5-2.8

2.3 Mobile Robot Abstraction

Figure 2.3 shows the differential drive robot abstraction we use in this research. It has a heading direction indicating that the robot knows which way it is headed. To detect obstacles, the robot is equipped with proximity sensors. Practical robots may use ultrasound sensors to detect obstacles, and a digital compass to calculate the heading. Each ultrasonic sensor is represented by beams emanating from the robot's center. They are labeled according to their relative positioning on the robot. For example US_LEFT45 indicates ultrasonic sensor at an angle 45^0 to the left of heading.

The robot navigation algorithm described in chapter 8, uses the navigation dynamics of differential drive robot described in this chapter.

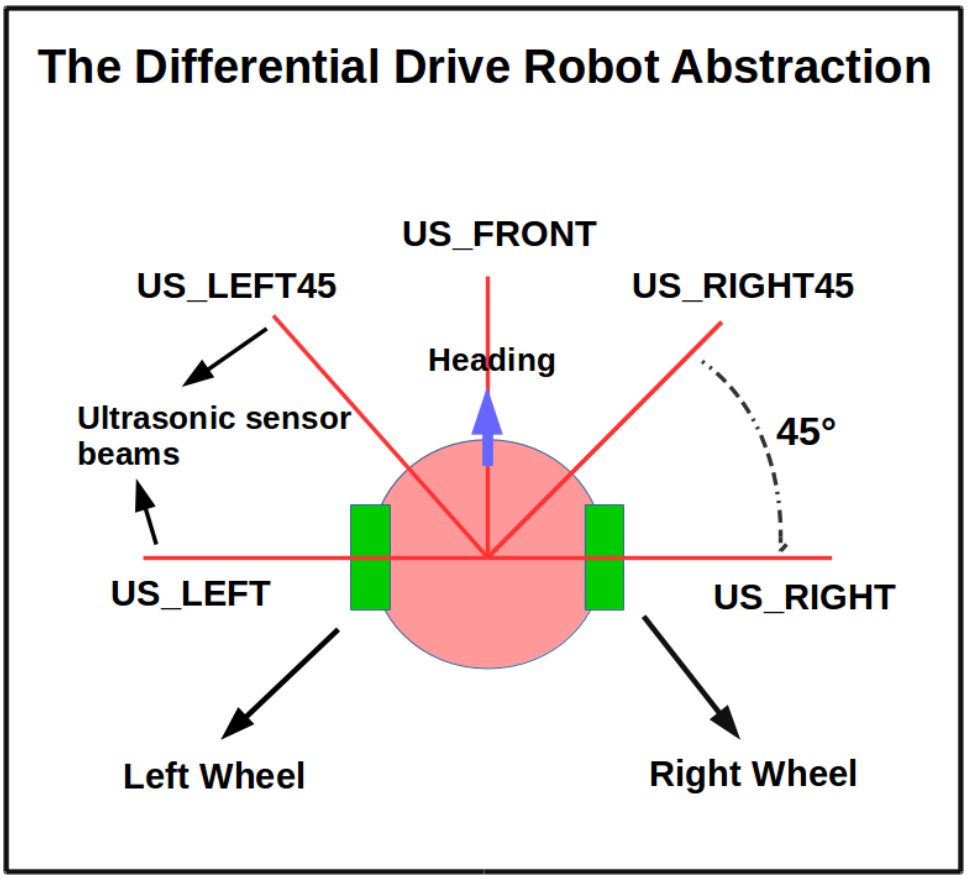

Figure 2.3: Differential drive robot model used in this research. Reprinted with permission from [1]. ©[2017] IEEE.

3. REINFORCEMENT LEARNING FRAMEWORK

In supervised learning [11], the machine learning model is presented with a correct output for every input given to it. So a supervised learning model can compare its prediction with the correct output for a given input and calculate an error metric over many such input examples. It then learns the predictive model by using an optimizing algorithm to minimize the error metric. In unsupervised learning [11], the model is presented only with the inputs. The correct outputs are not given to the model. The goal of unsupervised learning is to assign labels to input examples such that they can be formed into clusters with each cluster having a label of its own. The learning signal is present for every input in supervised learning, while it is totally absent in unsupervised learning. Between the two extremes is another form of machine learning, that presents the model with a sparse learning signal called reward. The learning model must associate this reward signal to its outputs and adapt its behavior so that it maximizes overall accumulated reward. This form of learning is called reinforcement learning.

Reinforcement learning is a machine learning framework used to solve problems involving a machine that can take actions in an environment unknown to it, to maximize a numeric reward provided to the machine by the environment [8]. The machine that takes actions is called an agent. The agent takes actions and finds itself in different situations called states in the environment. Upon taking an action, the agent is given a numeric reward signal by the environment that indicates the degree of to which the action is good for the agent. So the agent learn its sequence of actions using a trial and error method to eventually learn a set of actions that maximize its reward in the long run. To put simply, reinforcement learning is an intelligent way of doing trial and error learning.

Figure 3.1 illustrates the key components of reinforcement learning framework.

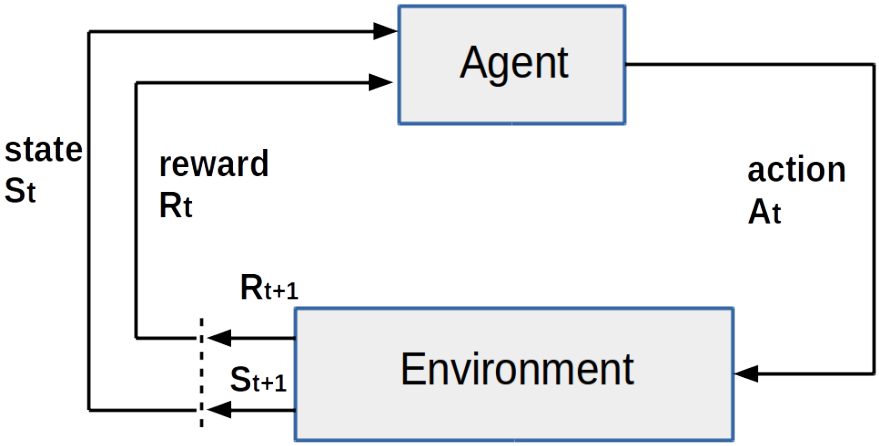

Figure 3.1: Agent-Environment interface in reinforcement learning. Adapted with permission from [8].

Thus any reinforcement learning problem can be identified by these components [8] :

1. **Agent :** A machine that interacts with its environment to learn actions to maximize a reward signal.

2. **Environment :** Everything that the agent interacts with, is comprised in the environment.

3. **State :** At every time step t of learning, the agent receives a representation of environment's state $s_t \in S$, where S is a set of all possible states.

4. **Action :** In a state s_t, the agent takes an action $a_t \in A(s_t)$, where $A(s_t)$ is a set of all actions the agent can take in state s_t

5. **Reward :** Upon taking an action a_t in state s_t, the agent is presented with a reward $r_{t+1} \in R$, where R is a set of possible rewards the environment can give to the agent. As a consequence of its action a_t in state s_t, the agent enters a new state s_{t+1}

Each learning episode is a sequence of time steps in which the agent visits states by taking actions, and may receive rewards for its actions. For our purpose we assume that an episode ends in a terminal state s_T in the final time step T for that episode. The notion of overall reward that the agent tries to maximize can be formalized by defining a return R_t [8]:

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + ... + r_T \tag{3.1}$$

where T is the last time step.

Inorder to make an agent perform a task using reinforcement learning framework, we must identify the environment, define state representation for the environment, and carefully design the reward function. The agent then accomplishes the task by maximizing its overall reward given by equation 3.1

A reinforcement learning framework requires that the state signal satisfy certain property to design learning algorithms. This is called the Markov property [8] discussed next.

3.1 The Markov Property

In a sequence of time steps involving agent environment interaction, we can observe that the state s_{t+1} entered by the agent at time step $t+1$ is dependent on the sequence of actions starting from state s_0. As such we can define the dynamics of the agent-environment interaction using the probability distribution [8] :

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ..., r_1, s_0, a_0\} \tag{3.2}$$

for all s', r, and for all possible values $s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ..., r_1, s_0, a_0$. Markov property dictates that the state of the agent at time step $t + 1$, and the reward received r_{t+1} only depends on the previous state s_t and action taken a_t, that is we can represent the

dynamics using just [8] :

$$Pr\{s_{t+1} = s', r_{t+1} = r|s_t, a_t\} \tag{3.3}$$

Using the Markov property, one can design algorithms to predict the reward r_{t+1} and state s_{t+1} using just one previous step s_t and a_t.

The next section discusses Markov Decision Processes (MDP) to formalize the usage of Markov property in designing reinforcement learning algorithms.

3.2 Markov Decision Processes

A task that can be solved using reinforcement learning and satisfies Markov property is called a Markov decision process (MDP). In addition to the components of reinforcement learning discussed in this section, an MDP is characterized by the one step dynamics for the next state and expected next step reward [8] :

$$p(s'|s, a) = Pr\{s_{t+1} = s|s_t = s, a_t = a\} \tag{3.4}$$

$$r(s, a) = E[R_{t+1}|S_t = s, A_t = a, S_{t+1} = s'] \tag{3.5}$$

Equation 3.4 represents the state transition probability and 3.5 represents the expected value of future reward in the next time step.

3.2.1 Policy Function

Policy function is defined for a Markov state to map actions to states with a probability. It defines the probability with which an action a is taken at state s [8].

$$\pi(s, a) = Pr\{A_t = a|S_t = s\} \tag{3.6}$$

3.2.2 Value Function

Value function is defined for a Markov state to estimate how good the state is for the agent. It defines value of a state s if a policy π is followed thereafter. Therefore it must reflect the expected value of rewards the agent can get from that state [8]

$$V^{\pi}(s) = E_{\pi}\{R_t | s_t = s\} \tag{3.7}$$

To design a reinforcement learning algorithm, the value and policy functions are defined for the task, and are evaluated using iterative techniques like policy iteration and value iteration [8]

In chapter 7, a way to implement and use the value and policy functions is presented.

Now, a deeper look into the key challenges in achieving goals using reinforcement learning is presented.

3.3 Key Challenges in Reinforcement Learning

A reinforcement learning agent must evaluate actions using a trial and error approach. As such it has two choices in every state : exploration and exploitation.

3.3.1 Action Selection

1. **Exploitation** : The learning agent utilizes existing knowledge of values of states to select an action. In exploition phase, an agent takes action that maximizes the value in that state.

2. **Exploration** : In the exploration phase the learning agent selects an action at random, to explore the value of that action in a given state.

The two phases are carried out by agent using an ϵ greedy policy. That is it can undergo exploitation with a probability ϵ and exploration with a probability $1 - \epsilon$.

3.3.2 Delayed Reward Problem

In a reinforcement learning task, a numeric reward may be given to an agent only after a few time steps. So there is a delay between taking an action and getting a reward. The agent must associate its reward with this prior actions. This is called the "delayed reward" problem or "credit assignment" problem.

In chapter 6, the navigation problem we consider in this research is shown to have the delayed reward nature. Chapter 5 discusses how spiking neural networks naturally solve the delayed reward problem. In chapter 7, we integrate the ideas in chapter 6 and 7 to arrive at the research goal that is to navigate the mobile robot to target in shortest time path.

4. SPIKING NEURONS AND SPIKING NEURAL NETWORKS *

In the recent years, artificial neural networks (ANN), also called the second generation neural networks have revolutionized the way we do machine learning. They have shown high accuracies in image recognition, speech recognition, natural language processing tasks. The traditional ANNs use backpropagation algorithm [12] to train the network. While artificial neural networks take their inspiration from the way biological neurons work, they do not tend to utilize the same learning mechanisms as the biological brains. To understand how the biological brains work and to take advantage of the computational power of biological neurons, a third generation of neurons is considered. They are called spiking neurons. The networks of spiking neurons are called spiking neural networks.

Spiking neurons model real biological neurons and are so called because they interact with each other using spikes in their membrane potential. Figure 4.1 illustrates the structure of a biological neuron. The neuron cell body has a negative potential with respect to its surroundings. A spike, also called action potential is produced if this membrane potential exceeds a threshold value. A neuron receives spikes from other neurons via dendrites. A generated action potential spike travels through the axon and is transmitted through the synaptic terminals to other neurons' dendrites.

The action potential spike transmission is visualized in Figure 4.2

4.1 Structure of Action Potential Spike

The graphical representation of a typical action potential spike with respect to time is shown in Figure 4.3

When a neuron receives external stimulus from dendrites, its membrane potential rises

*Parts of the material presented in this chapter are reprinted with permission from "Simple model of spiking neurons" by E.M. Izhikevich. *IEEE transactions on Neural Networks*, vol. 14, pp. 1569-1572, Nov 2003. ©[2003] by IEEE.

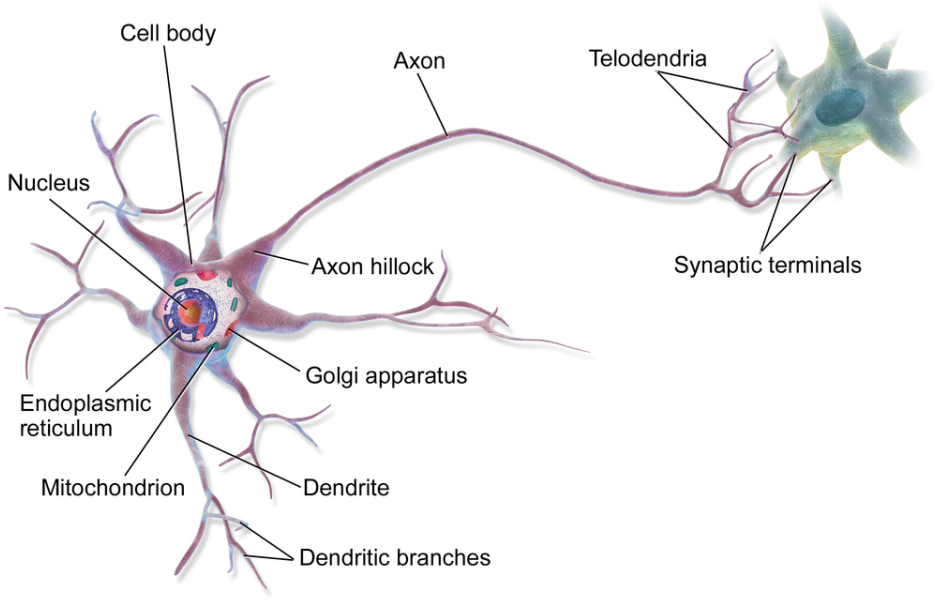

Figure 4.1: Structure of a biological neuron by Bruce Blaus. Reprinted from [13].

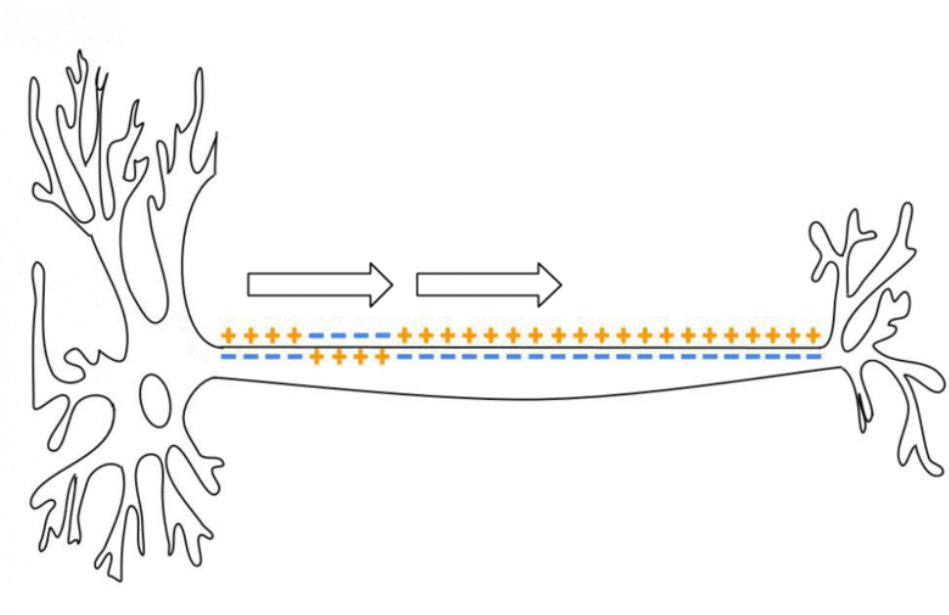

Figure 4.2: Transmission of action potential spike. Reprinted from [14].

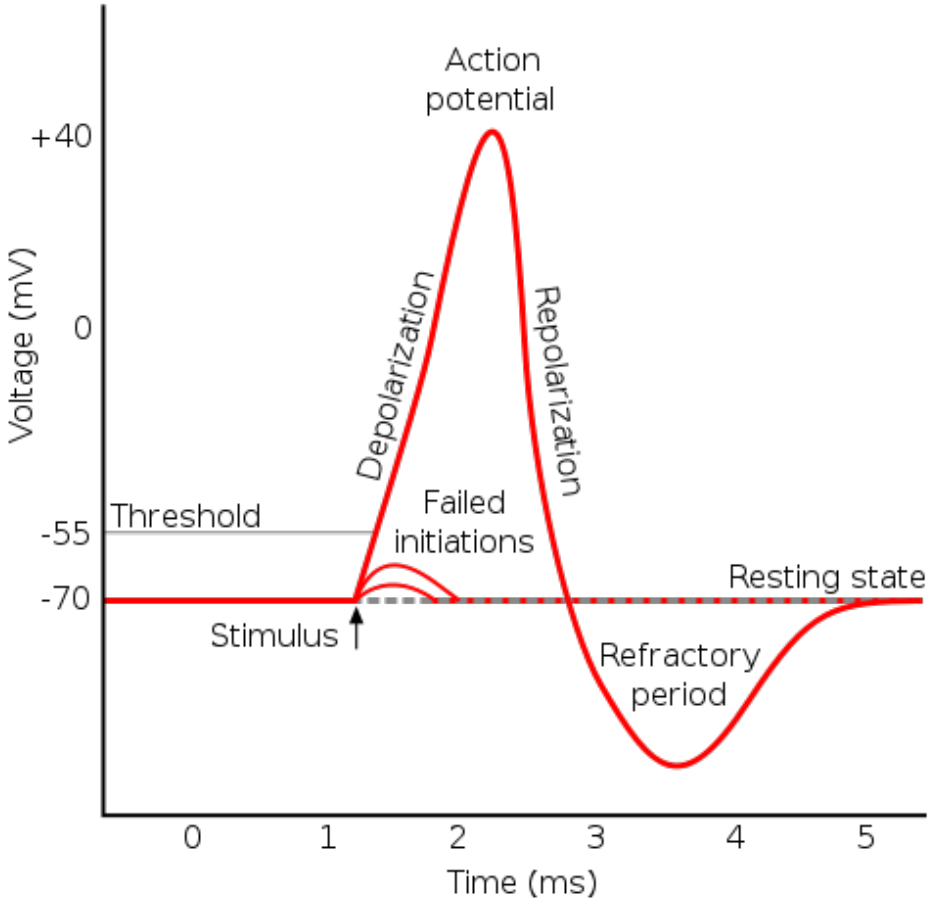

Figure 4.3: Structure of a spike. Reprinted from [15].

from its resting value (typically -70 mV). This process is called depolarization. When the membrane potential rises beyond the threshold value, it increases abruptly and then decreases causing a spike. Before the membrane potential reaches its resting state again, it goes into a refractory period to potential values lower than the resting potential. It then rises until it reaches the resting state.

4.2 Spiking Neuron Model

A typical approach to understand how a system works is to build a model of it and conduct experiments with the model. This can also be applied to spiking neurons. Many

models of biological spiking neurons have been created ranging from the extremely complex Hodgkin-Huxley model [16] to a simple leaky integrate and fire model [17]. The complex Hodgkin-Huxley model of spiking neuron is computationally intensive to use in simulations that do not require such detailed biological realism. On the other hand leaky integrate and fire (LIF) models are computationally efficient and can be used if the simulation requires just the spike patterns rather than the exact spike structure.

In this research we use Izhikevich model [18] that lies somewhere between Hodgkin-Huxley and LIF model interms of computational complexity and biological realism. The dynamics of Izhikevich model are discussed next.

4.2.1 Izhikevich Model

Izhikevich spiking neuron model uses a 2-dimensional system of ordinary differential equations to describe the neuron dynamics [18]. If v is the action potential of the neuron membrane, I is the injected external current into the membrane, then Izhikevich model is described by [18] :

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{4.1}$$

$$\frac{du}{dt} = a(bv - u) \tag{4.2}$$

The spike resetting is done using these conditions:

$$if\ v \geq 30\ mV,\ then\ v = c\ and\ u = u + d \tag{4.3}$$

In these equations, u represents a membrane recovery variable. According to [18], the parameter descriptions are :

1. Membrane potential v in the model lies between -70 mV and -60 mV depending on

b .

2. Threshold potential is between -55 mV and -40 mV.

3. Parameter a affects the speed of recovery. Smaller the value of a, slower is the recovery. Typically $a = 0.02$.

4. Parameter b defines the sensitivity of recovery variable u to changes in v. Typically $b = 0.2$

5. Parameter c defines the reset value of the membrane potential v after the spike. Typically $c = -65\ mV$

The membrane potential variation v/s time resulting from equations 4.1 - 4.3 is shown in Figure 4.4

4.3 Spiking Neuron Abstraction

In order to perform computations with spiking neurons, we consider the spike patterns produced by a spiking neuron rather than the exact shape of the spike. The relative timing between the spikes and rate of spikes generated are the desired attributes of a spiking neuron. Therefore when discussing behavior of a network of neurons, it becomes easier to consider a spike as a finite amplitude delta function rather than a spread out action potential curve. This spike timing based view is shown in Figure 4.5

4.4 Spiking Neural Networks

A network of spiking of neurons is simply a group of spiking neurons connected by synaptic weights. A simple spiking neural network is shown in Figure 4.6. The synaptic weights are denoted by W_{11} connecting input neuron 1 to output neuron 1, and W_{12} connecting input neuron 2 to output neuron 1. For a given synapse, presynaptic neurons are the ones that transmit their spikes via the synapse. Postsynaptic neurons are the ones that

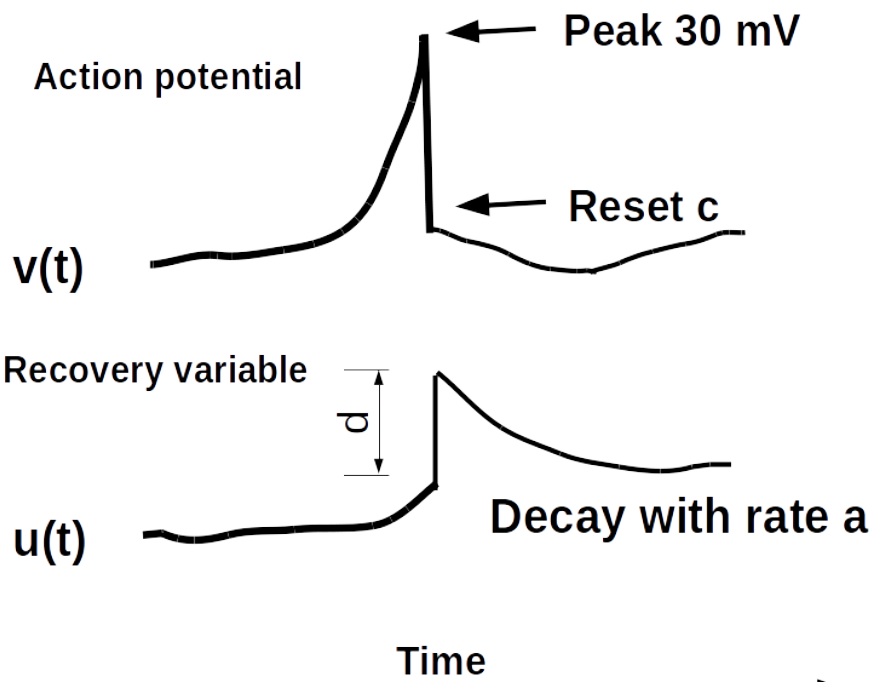

Figure 4.4: Spike generation with Izhikevich model. Adapted with permission from [18]. ©[2003] IEEE.

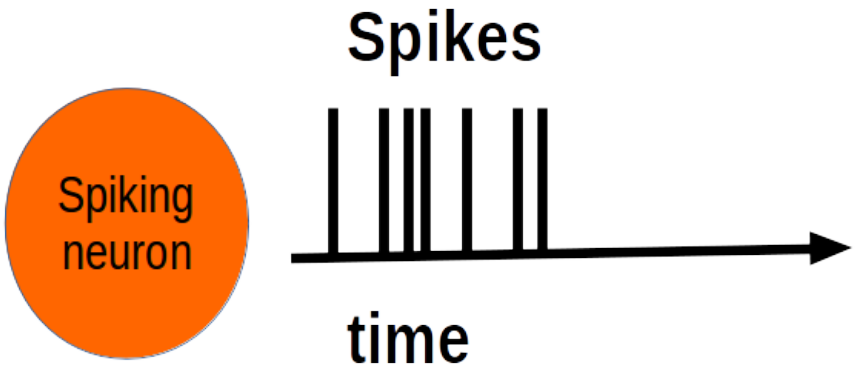

Figure 4.5: Spiking neuron activity represented by discrete time spikes

receive the spikes from presynaptic neurons. A synapse converts spikes into currents that are injected into postsynaptic neurons.

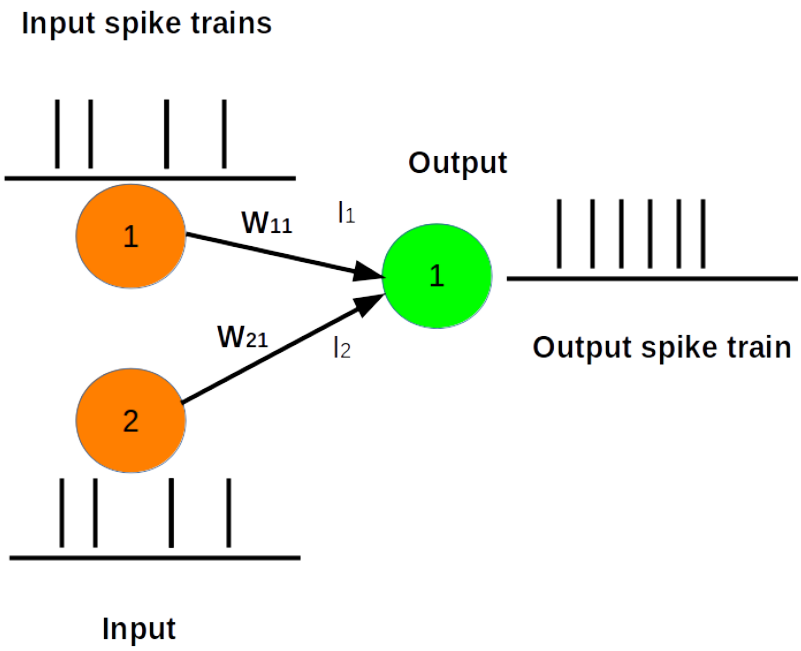

Figure 4.6: Simple spiking neural network

In a network of spiking neurons such as the one in Figure 4.6, a neuron j is injected with currents I_{ij} from presynaptic neurons i calculated as follows :

$$I_{ij} = \sum_{t=0}^{T} S_{ij}(t) W_{ij} \tag{4.4}$$

Here, $S_{ij}(t) = 1$ if neuron i outputs a spike at timestep t.

4.5 Information Encoding in Spiking Neural Networks

As shown in section 4.4, spiking neurons communicate with each other using a set of spikes. This requires us to use an encoding scheme to interpret the spike patterns. Two such encoding methods are known:

1. Rate encoding : Spikes are interpreted by their frequency. If a neuron outputs n_s number of spikes occur in time T, then the rate based encoding indicates that the output of this neuron is proportional to n_s/T

2. Temporal encoding : Interspike interval time can take any values. Therefore one can also use interspike intervals to differentiate between spike patterns.

In this research, rate encoding scheme is used.

4.6 Learning in Spiking Neural Networks

As indicated by equation 4.4, the output of a spiking neuron is influenced by its incoming synaptic weights. Learning in spiking neural networks should thus involve synaptic weight changes. Biological neurons utilize a mechanism called Hebbian spike timing dependent plasticity (STDP) [19] to change synaptic weights.

4.6.1 Spike Timing Dependent Plasticity

Consider a simple spiking neural network with a single synapse connecting a presynaptic neuron i to a postsynaptic neuron j as shown in Figure 4.7. Let the synaptic strength be denoted by W_{ij} . If t_{post} is any spike time of postsynaptic neuron, t_{pre} is any spike time of presynaptic neuron, then the synaptic weight change happens according to Hebbian STDP [19] as shown below :

$$STDP(\Delta t) = A_+ exp(-\frac{\Delta t}{\tau_+}) \text{ for } \Delta t > 0 \tag{4.5}$$

$$STDP(\Delta t) = A\_exp(-\frac{\Delta t}{\tau_-}) \text{ for } \Delta t <= 0 \tag{4.6}$$

where $\Delta t = t_{post} - t_{pre}$. A_+, A_-, τ_+, τ_- are design parameters. A graphical representation of the STDP is shown in Fig 4.7 labeled "STDP Curve".

$STDP(\Delta t)$ represents the weight change caused by presynaptic and postsynaptic spike pair with timings t_{pre} and t_{post}. If the spike activity happens for a time interval T, and if T_{PRE} represents the set of presynaptic spike times, T_{POST} represents the set of postsynaptic times, then the total STDP weight change ΔW_{ij} for a synapse W_{ij} is given by :

$$\Delta W_{ij} = \sum_{t_{pre}} \sum_{t_{post}} STDP(t_{post} - t_{pre}) \tag{4.7}$$

where $t_{pre} \in T_{PRE}$ and $t_{post} \in T_{POST}$. Equation 4.5 leads to increase in synaptic weight W_{ij} and equation 4.6 leads to decrease in synaptic weight. The phenomenon described by equation 4.5 is called "long term potentiation" (LTP) and that described by equation 4.6 is called "long term depression" (LTD). Therefore LTP happens if a post synaptic spike occurs after a presynaptic spike, and LTD happens if a pre synaptic spike occurs after a post synaptic spike. In other words, LTP represents correlation between presynaptic and postsynaptic spike activity while LTD represents anti-causal relationship between presynaptic and postsynaptic activity.

4.6.2 Nearest Neighbor STDP

The definition of STDP in equations 4.5 - 4.6 do not put any restriction on the number of pre-post synaptic spike pairs to consider to calculate $\Delta t = t_{post} - t_{pre}$. It is computationally inefficient to consider all possible pairs of presynaptic and postsynaptic spikes. Therefore we use a simplistic version of STDP called the "nearest neighbor STDP". It is illustrated in Figure 4.8

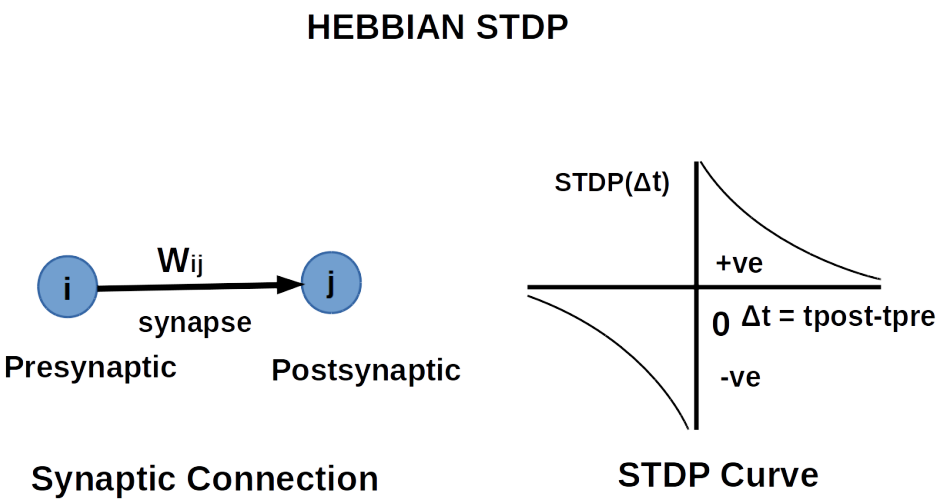

Figure 4.7: Synaptic Connection and STDP curve. Adapted with permission from [1]. ©[2017] IEEE.

In Figure 4.8, a presynaptic spike is paired with only one most recent postsynaptic spike to cause LTD. Similarly a postsynaptic spike is paired with only one most recent presynaptci spike to cause LTP.

A biologically realistic learning algorithm must use some form of STDP mechanism for synaptic weight changes. Chapter 5 discusses the implementation of reinforcement learning using spiking neural networks by utilizing a modulated form of STDP.

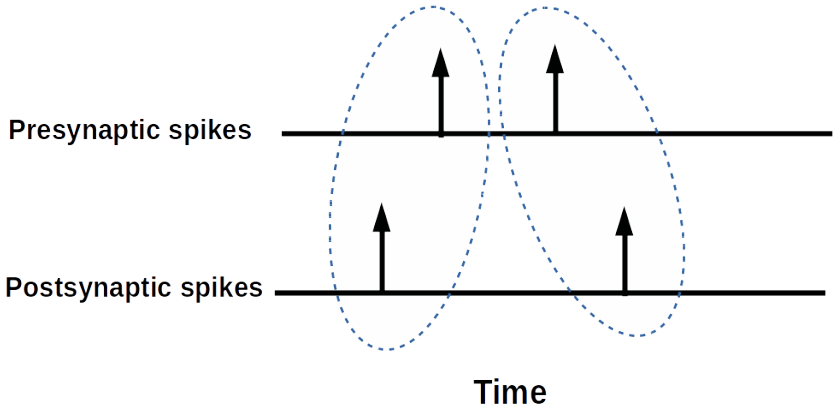

Figure 4.8: Nearest neighbor STDP

5. REINFORCEMENT LEARNING WITH SPIKING NEURAL NETWORKS *

Learning from interaction is a fundamental aspect of animal behavior. Take humans for instance. We understand several things about our environment on our own by interacting with it. When an infant touches a cup of hot beverage, it associates the sensation of heat to its action of touch the cup and learns to be careful with hot cups in future. The heat served as a learning signal for the infant. When training a dog, we do not explicitly teach the dog the human language. Instead we issue commands to the dog, and when the dog randomly chooses to correctly act on a command, we present it with a reward. The dog then links the reward with its behavior and tends to repeat it. This ability of animals to influence their environment by their actions in order to obtain maximum rewards is something that is developed over the course of evolution. As with any other animal behavior, this reward seeking behavior is caused by its brain. As such we might suspect that the underlying building blocks of brains that is the spiking neurons are responsible for this reinforcement learning behavior. This chapter presents a possible way the spiking neurons might implement reward seeking behavior. First let us look at a phenomenon describing reinforcement learning behavior in animals : operant conditioning.

5.1 Operant Conditioning

Operant conditioning is a process in which an animals behavior in response to a stimulus is strengthened using a reward signal and weakened using punishment signal [20]. This is the process dog trainers use to train them.

*Parts of the material presented in this chapter are reprinted with permission from "Navigating mobile robots to target in near shortest time using reinforcement learning with spiking neural networks" by Amarnath Mahadevuni and Peng Li, 2017. *International Joint Conference on Neural Networks(IJCNN)*, pp.2243-2250, May 2017. ©[2017] by IEEE, and "Solving the distal reward problem through linkage of STDP and dopamine signaling" by E.M. Izhikevich, 2007. Cerebral Cortex, Oxford University Press, Jan 2007. ©[2007] by Oxford University Press.

In the context of the brain, let us consider two connected layers of spiking neurons in which the first layer represents the stimulus, the second layer represents the response. The synaptic connections between the two layers determine how effectively a stimulus results in a response. This is illustrated in Figure 5.1 :

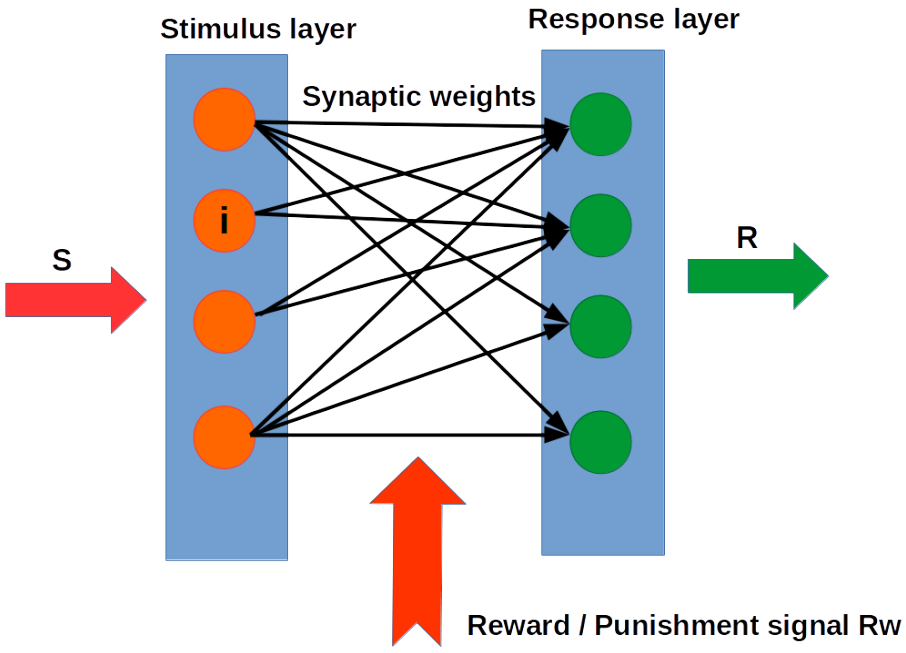

Figure 5.1: Simple Spiking Neural Network with Stimulus Layer and Response Layer

In Figure 5.1, let us assume that for a stimulus S, the neural network gives a response R. The stimulus S is converted into spike activity by the stimulus layer. To this network, if we apply a reward Rw, then according to operant conditioning, its response for the same stimulus S should be larger than R. Similarly if we apply a punishment Rw, then its response for the same stimulus S should be smaller than R. We have seen in chapter 4 that the postsynaptic response of a neural network is affected by the strength of the

synapses. Therefore it can be hypothesized that the reward or punishment signal should affect the synaptic weights in such a way that the response changes in accordance with operant conditioning. The chemical that can influence the synaptic weight changes in the biological brain is called a neuromodulator. The neuromodulator associated with reward in our brains is the dopamine [21] . Punishment can be considered as reduction in dopamine level.

A fundamental characteristic of operant conditioning is that the reward or punishment is typically applied several time steps after the brain elicits a response. By this time, the stimulus that caused the response may not even be there. This is also explained in the context of mathematical framework of reinforcement learning in chapter 3 as the "delayed reward problem". To handle this, [21] proposes a variable "eligibility trace" that memorizes the synaptic activity that led to a response. The next section discusses the use of eligibility trace in solving the delayed reward problem.

5.2 Reinforcement Learning with Dopamine Modulated STDP and Eligibility Trace

Consider a single synapse with a postsynaptic neuron j and presynaptic neuron i with synaptic strength W_{ij} in Figure 5.2 . Assume there is some stimulus applied to neuron i

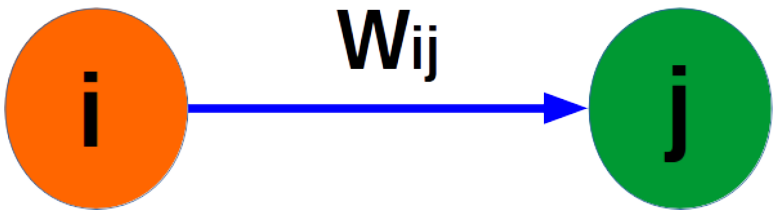

Figure 5.2: Simple Synaptic Connection with Synaptic Weight W_{ij}

which generates a presynaptic spike activity, and the synaptic connection causes a postsy-

naptic spike activity in neuron j. This is shown in Figure 5.3

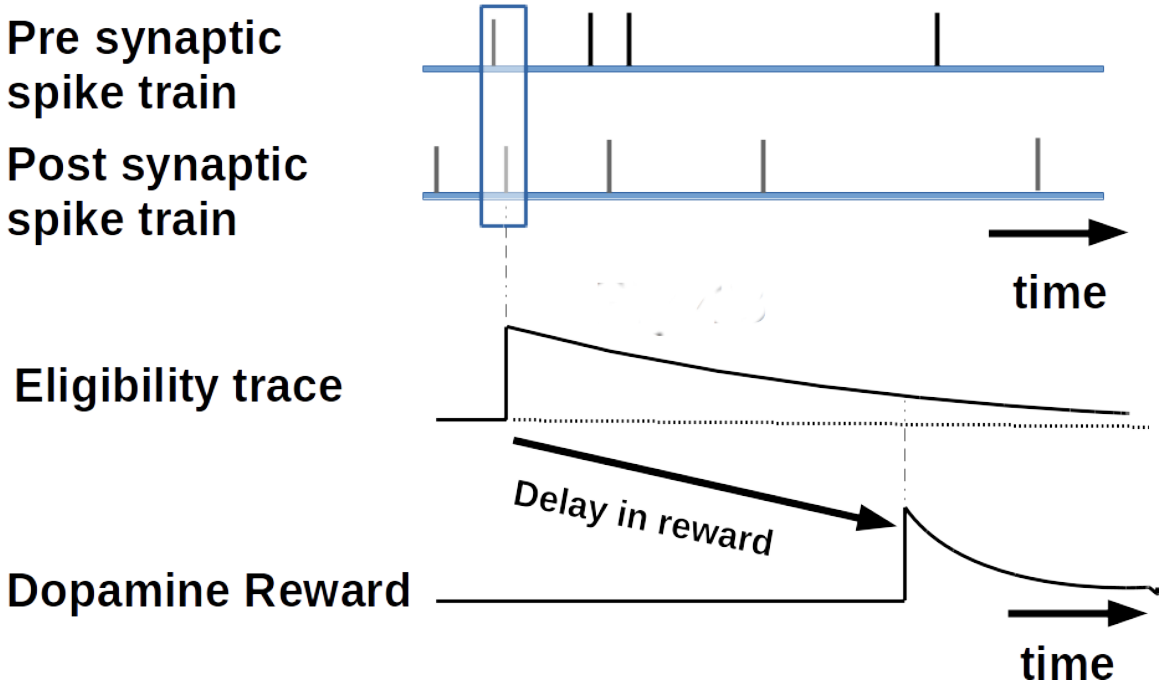

Figure 5.3: Pre and post synaptic activity with eligibility trace and dopamine reward signal. Adapted with permission from [21]. ©[2007] Oxford University Press.

Eligibility trace shown in Figure 5.3 is a variable that remembers the activity that resulted in an STDP weight change (calculated using 4.7). It is the STDP weight change decayed over time so that its relevance is reduced as time passes by. When a reward is applied in the form of dopamine, the numeric reward value is multiplied with the decayed eligibility trace to form the effective synaptic weight change. The decay of eligibility trace every timestep is done as follows:

$$\Delta W_{ij}(t+1) = \Delta W_{ij}(t) * \beta \tag{5.1}$$

where $0 < \beta < 1$ is the factor by which eligibility trace is decayed every time step. Therefore if STDP weight change is calculated at time t_0, then after n time steps, the eligibility trace is given by:

$$\Delta W_{ij}(t_0 + n) = \Delta W_{ij}(t_0) * \beta^n \tag{5.2}$$

Since $\beta < 1$, 5.2 decays the eligibility trace over time. The synaptic weight change is applied to the synapse only when the reward is applied. If the reward is applied after m time steps, then the synaptic weights are changed as follows:

$$W_{ij} = W_{ij} + \Delta W_{ij}(t_0 + m) * reward * \delta(t - t_{reward}) \tag{5.3}$$

$$= W_{ij} + \Delta W_{ij}(t_0) * \beta^m * reward * \delta(t - t_{reward}) \tag{5.4}$$

where δ is a unit amplitude delta function, and t_{reward} is the time when reward is applied.

5.3 Designing Spiking Neural Network Architecture for Reinforcement Learning

Chapter 3 discusses that in order to design solutions to reinforcement learning problems, we must first identify agent's states, actions, and rewards for the problem. Then encode them using some representation. In spiking neural networks, one can use a layer to represent states, a layer to represent actions, and dopamine modulation to represent reward. A typical reinforcement learning implementation with spiking neural network is shown in Figure 5.4

Next three subsections explain the roles of these layers and dopamine signal.

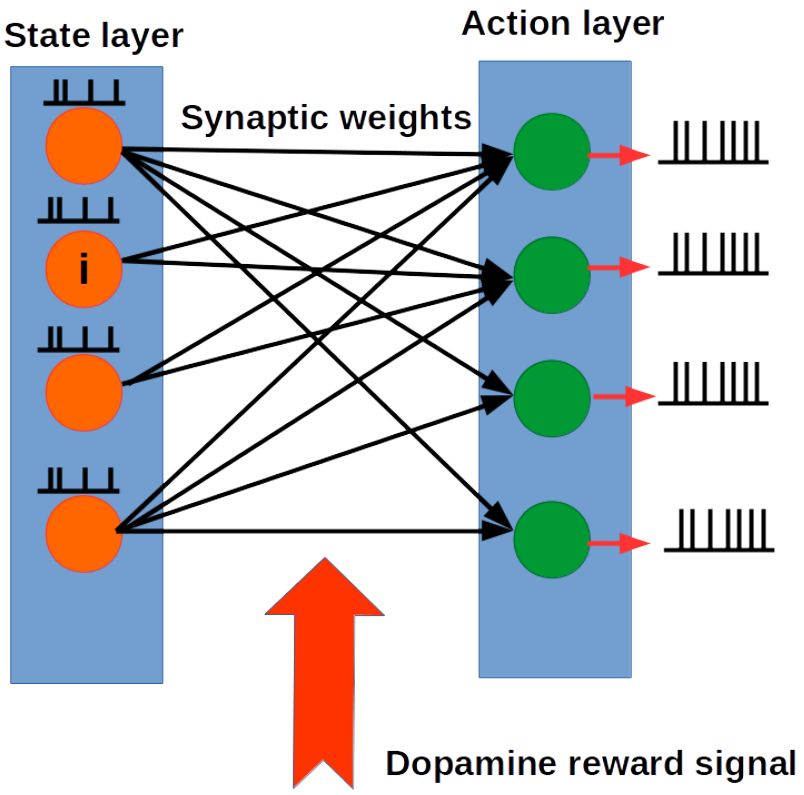

Figure 5.4: Simple Spiking Neural Network Architecture to Implement Reinforcement Learning.

5.3.1 State Layer

The state layer is the stimulus layer that converts a state representation into spike frequencies to generate a spike train. For example a function $f(S)$ can be applied on a state S to generate a frequency f_i for each neuron i in the input layer.

5.3.2 Action Layer

At a given state, the input state layer neuron i undergoes a spike activity depending on its corresponding spike frequency f_i. This results in spike activity in action layer. The

spike frequency can be interpreted as value of an action for the state S .

5.3.3 Dopamine Reward Signal

The reward function can be designed for the goal at hand, and a dopamine reward signal can be used to modulate synaptic weights connecting state layer and action layer to affect the agent's response to a state.

Chapter 7 uses this principle to design a spiking neural network architecture for the navigation problem in this research.

6. PREVIOUS WORK*

Consider the 2-D environment shown in Figure 6.1 also discussed in chapter 1.

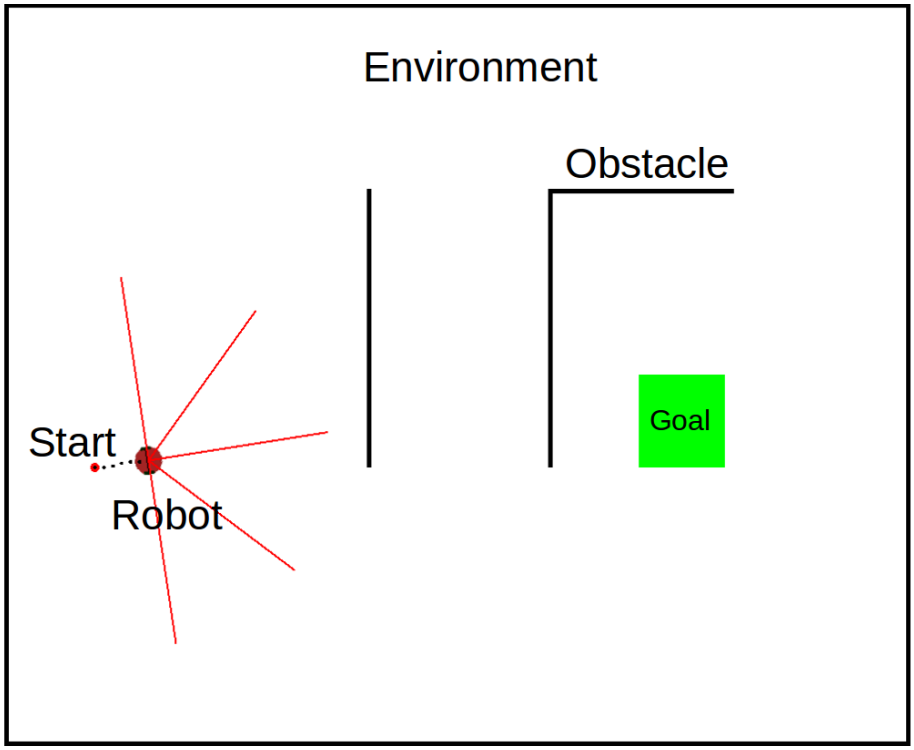

Figure 6.1: 2-D environment for the navigation problem. Reprinted with permission from [1]. ©[2017] IEEE.

In this chapter, an existing solution [22] to problem of mobile robot navigation to a goal location in the 2-D environment of Figure 6.1 is discussed. It is a state machine based approach [22]. It does not consider optimizing the navigation time to a goal. In this document, this solution is referred to as "go-to-goal" approach.

*Parts of the material presented in this chapter are reprinted with permission from "Navigating mobile robots to target in near shortest time using reinforcement learning with spiking neural networks" by Amarnath Mahadevuni and Peng Li, 2017. *International Joint Conference on Neural Networks(IJCNN)*, pp.2243-2250, May 2017. ©[2017] by IEEE.

6.1 Go-to-goal Solution

6.1.1 Wall Following Behavior

The go-to-goal approach depends on a robotic behavior called wall following where the robot moves alongside a wall-like obstacle by maintaining a constant distance with it. During wall following, the obstacle can be to the left or right side of the robot heading and thus leads to two behaviors : WALL-FOLLOW-LEFT and WALL-FOLLOW-RIGHT. Figures 6.2 and 6.3 shows these behaviors.

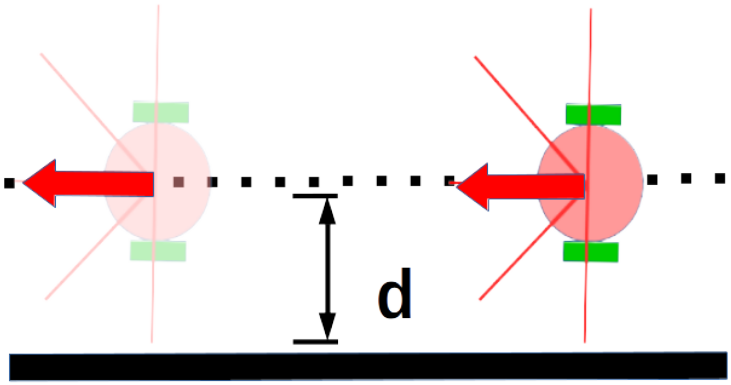

Figure 6.2: WALL-FOLLOW-LEFT.

1. WALL-FOLLOW-LEFT : The mobile robot maintains a constant distance d from the obstacle with the obstacle on left side of its heading direction. See Figure 6.2.

2. WALL-FOLLOW-RIGHT : The mobile robot maintains a constant distance d from the obstacle with the obstacle on right side of its heading direction. See Figure 6.3.

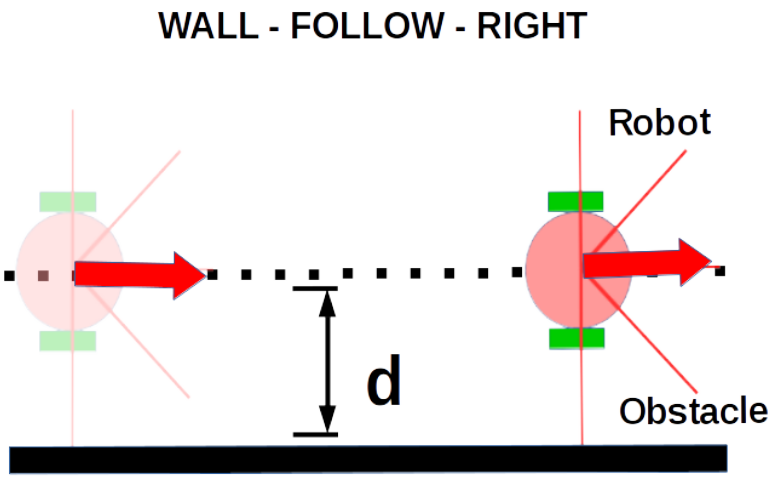

Figure 6.3: WALL-FOLLOW-RIGHT.

Go-to-goal uses the wall following behavior. Figure 6.4 shows the trajectory of the robot using this approach.

The trajectory of the robot can be summarized as follows:

1. Move towards goal when there is no obstacle in vicinity.

2. When faced with an obstacle, turn in place, then move in wall following manner.

3. Quit wall following when there is a clear path to goal.

Flow chart in Figure 6.5 shows an algorithmic view of the trajectory.

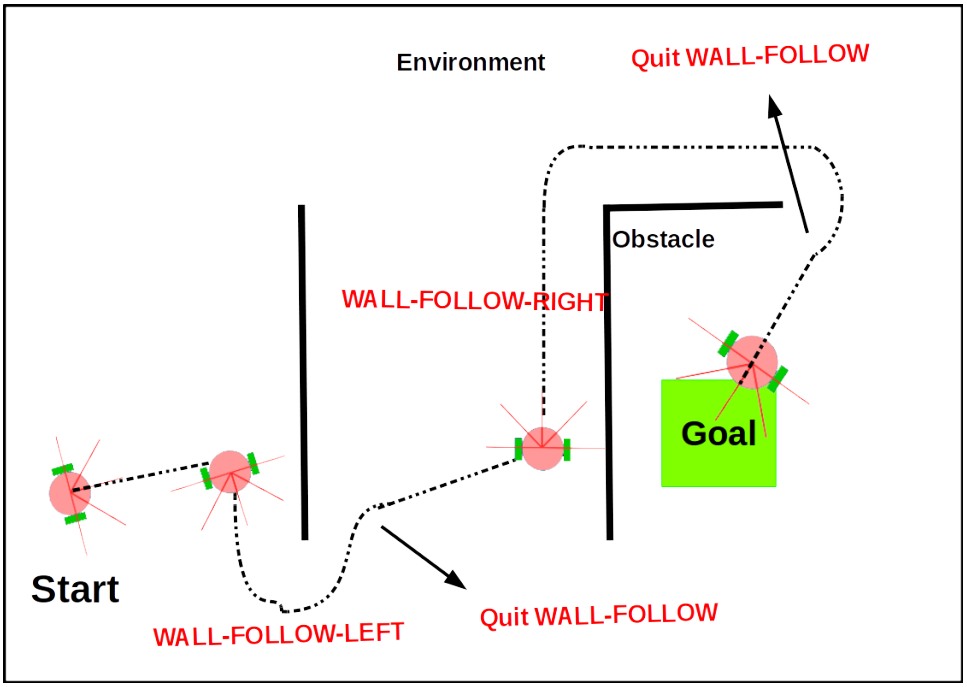

Figure 6.4: Go-to-goal solution.

6.2 State Machine Implementation of Go-to-goal

Chapter 7 discusses the solution proposed in this research. It is build on top of go-to-goal approach. As such, it is necessary to look at the implementation details of this solution. This helps in understanding the complete proposed solution in chapter 8. (Please keep in mind that the "state" referred to here is the navigation state and not related to the states in reinforcement learning framework discussed in chapter 3).

Figure 6.6 shows the go-to-goal trajectory split into several states.

The robot's movement is governed by principles described in 2. Its robot behavior in each of the states in Figure 6.6 is as follows [1]:

1. PRE-GO-TO-GOAL : Orients itself in place towards the goal. Then transitions to GO-TO-GOAL state.

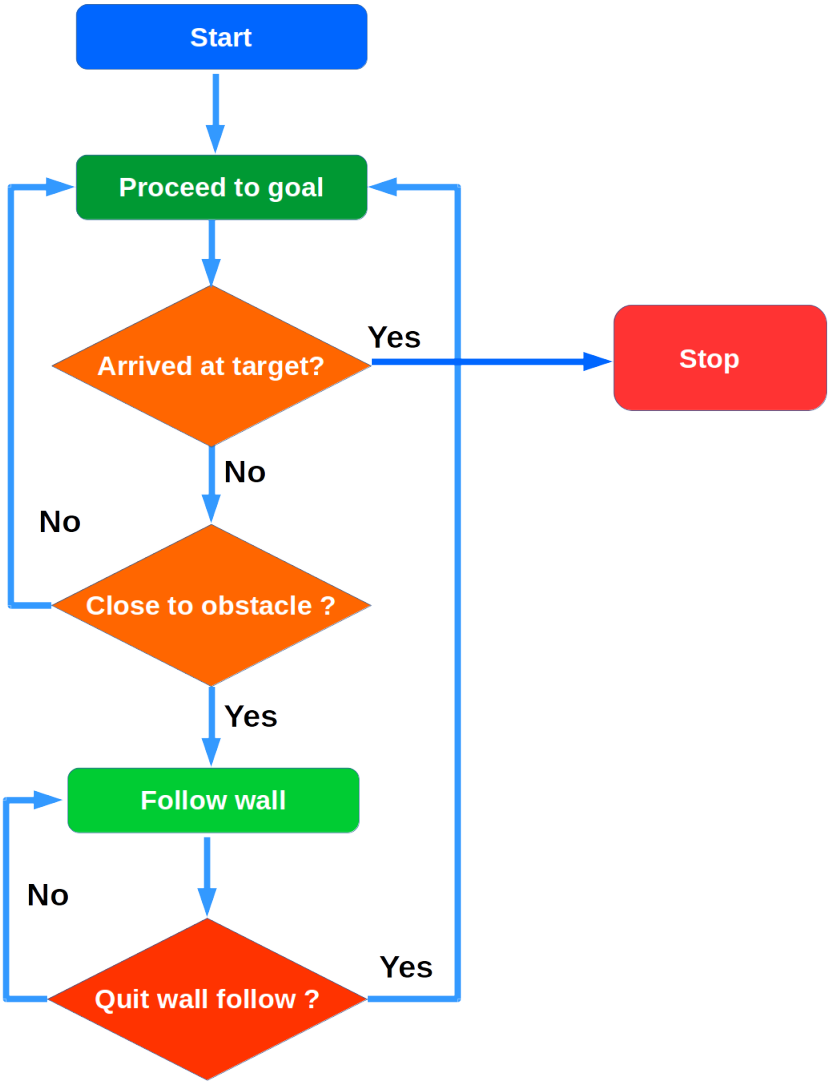

Figure 6.5: Flow chart depicting go-to-goal approach.

2. GO-TO-GOAL : Moves towards the goal if there is no obstacle close to it. If there is an obstacle close to it, the robot transitions to PRE-WALL-FOLLOW state.

3. PRE-WALL-FOLLOW : Turns in place to place itself parallel to the obstacle. Then transitions to WALL-FOLLOW state.

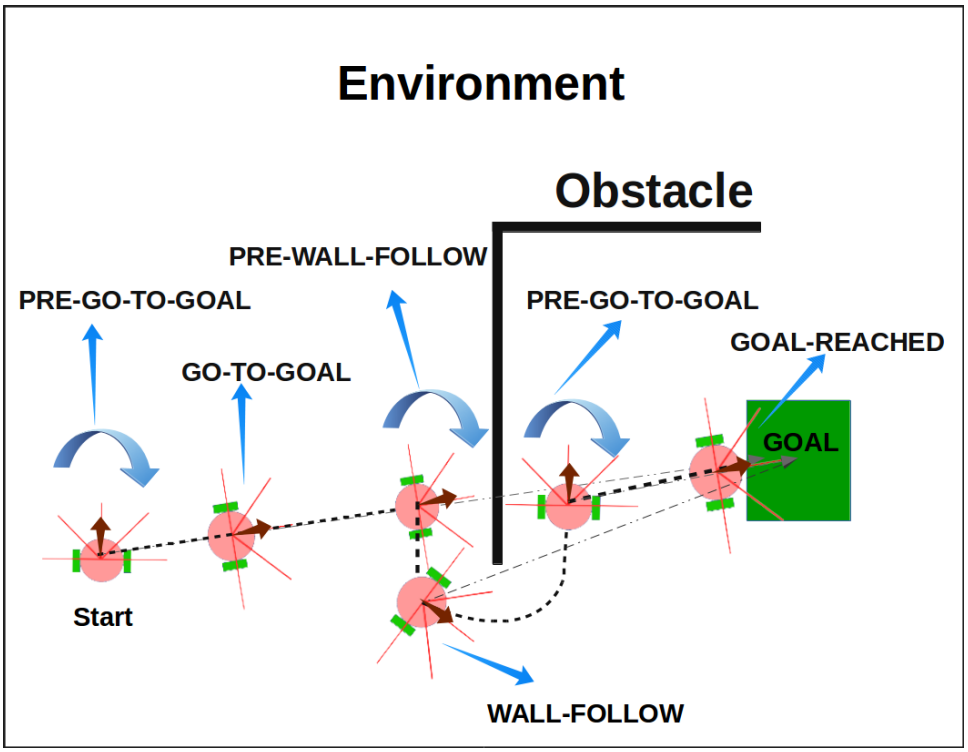

Figure 6.6: Go-to-goal trajectory split into navigation states. Reprinted with permission from [1]. ©[2017] IEEE.

4. WALL-FOLLOW : Proceeds to move in wall following manner. WALL-FOLLOW-LEFT or WALL-FOLLOW-RIGHT is adopted depending on the direction of turn in PRE-WALL-FOLLOW state. The robot checks for certain conditions to see if it can quit WALL-FOLLOW state and transition to PRE-GO-TO-GOAL state. This checking is described in section 6.2.1.

5. GOAL-REACHED : Arrives at goal location.

6.2.1 Conditions to Quit Wall Following Behavior

The following parameters are important to understand the conditions that the robot needs to check before quitting wall following behavior.

1. **Obstacle avoidance vector A** : Indicates the direction the robot needs to move if

it must avoid obstacles in its vicinity. In Figure 6.7, the obstacle Avoidance Vector **A** is calculated as a resultant of vectors with directions opposite to the ultrasonic beam direction and magnitude inversely related to the proximity from the obstacle. Equation 6.1 shows the calculations.

$$\begin{aligned}\mathbf{A} = \mathbf{V}_{US\_LEFT} * d(US\_LEFT) + \mathbf{V}_{US\_LEFT45} * d(US\_LEFT45) + \\ + \mathbf{V}_{US\_FRONT} * d(US\_FRONT) + \mathbf{V}_{US\_RIGHT45} * d(US\_RIGHT45) + \\ + \mathbf{V}_{US\_RIGHT} * d(US\_RIGHT) \quad (6.1)\end{aligned}$$

In equation 6.1, the vectors $\mathbf{V}_{<SENS>}$ represent unit vectors in the opposite to the direction of proximity sensor beams discussed in chapter 2, section 2.3, where $SENS \in$ *{US_LEFT, US_LEFT45, US_FRONT, US_FRONT45, US_RIGHT, US_RIGHT45}* and $d(< SENS >)$ are the proximities measured by sensor labeled $SENS$.

2. **Goal vector G** : The goal vector is simply a vector with tail at the center of the robot and head at the center of the goal.

3. **Angle** $\alpha_{(\mathbf{G},\mathbf{A})}$: The smaller angle between vectors **A** and **G**.

4. **Distance to target in PRE-WALL-FOLLOW state** $d_{pre-wall-follow}$: Distance from the robot's center to the goal location calculated before the robot transitions to WALL-FOLLOW state.

Wall following behavior is quit when both of these conditions hold :

1. $-90^0 < \alpha_{(\mathbf{G},\mathbf{A})} < 90^0$. If this condition is satisfied then we know that the goal and head of obstacle avoidance direction are on the same side of the robot.

2. $|\mathbf{G}| < d_{pre-wall-follow}$. If this condition is satisfied then we know that the robot is closer to the goal than when it begins following the wall.

If these conditions are met, the robot transitions from WALL-FOLLOW state to PRE-GO-TO-GOAL state.

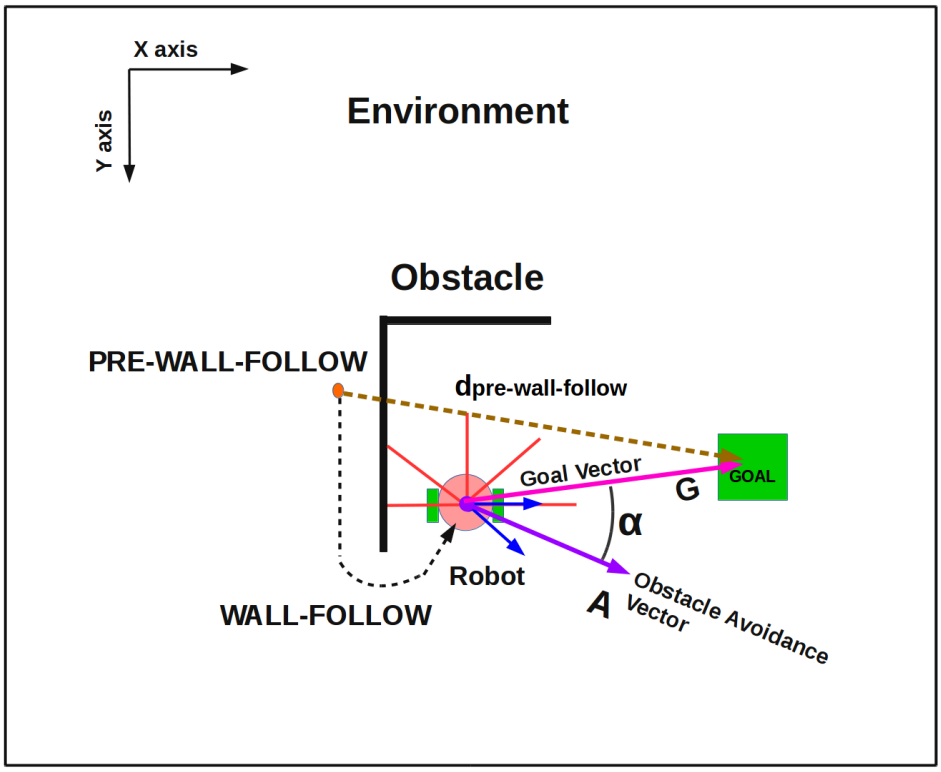

Figure 6.7: Parameter definitions related to the conditions to leave wall following behavior. Adapted with permission from [1]. ©[2017] IEEE.

7. FORMULATING THE REINFORCEMENT LEARNING PROBLEM

We shall first revisit the proposed research problem and then see its formulation as a reinforcement learning problem.

7.1 Minimizing the Navigation Time

Consider Figure 7.1 . Our goal is to minimize the time it takes for the robot to navigate from "Start" location to "Goal" location.

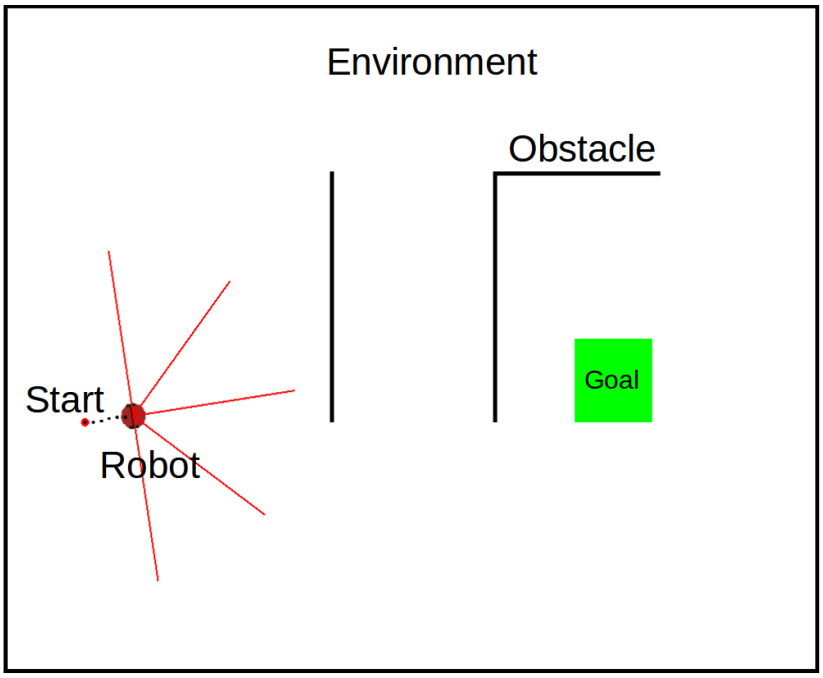

Figure 7.1: Example instance of the problem depicting the navigation goal. Reprinted with permission from [1]. ©[2017] IEEE.

If we apply the navigation solution "go-to-goal" to the problem instance in Figure 7.1, we get a trajectory shown in Figure 7.2.

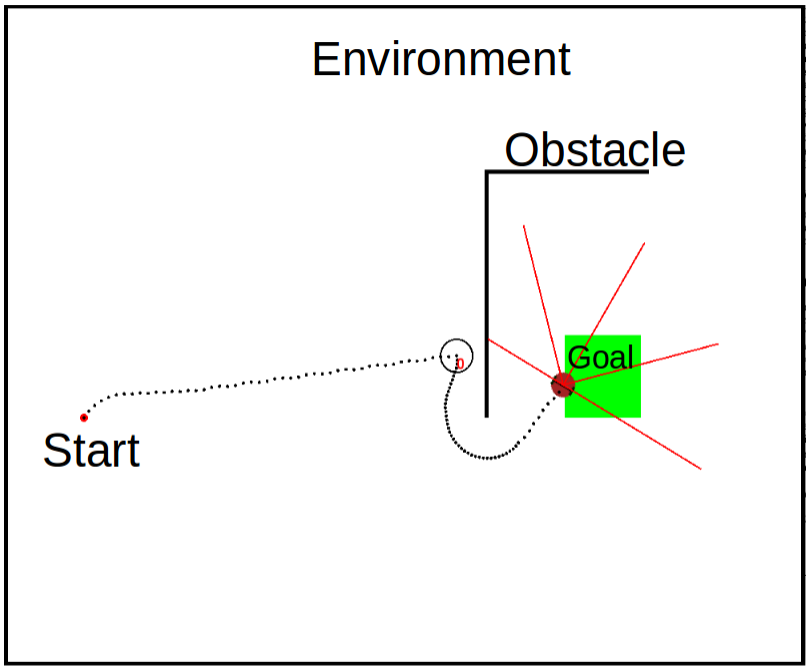

Figure 7.2: Go-to-goal solution applied to the setup in Figure 7.1. Reprinted with permission from [1]. ©[2017] IEEE.

The circled area of the trajectory is PRE-WALL-FOLLOW state. In this state, the robot can decide to follow the wall in one of two modes : WALL-FOLLOW-RIGHT and WALL-FOLLOW-LEFT. In the example in Figure 7.2 the robot decides to adopt a WALL-FOLLOW-LEFT behavior. It could have also chosen a WALL-FOLLOW-RIGHT behavior. The time taken by the robot to reach the goal when it selects WALL-FOLLOW-LEFT is smaller than the time taken when it selects WALL-FOLLOW-RIGHT. Thus the navigation time of the robot in "go-to-goal" solution depends on the wall following mode selected in the PRE-WALL-FOLLOW state.

The navigation time of the robot can therefore be minimized within "go-to-goal" solution by choosing the proper wall following mode at every PRE-WALL-FOLLOW state the robot encounters. The fact that the obstacles' location is unknown motivates us to use

a trial and error approach. Chapter 3 discusses that reinforcement learning is a good way to perform trial and error tasks. The next section shows the formulation of this problem of minimizing the navigation time as a reinforcement learning problem.

7.2 Framing the Problem as a Reinforcement Learning Task

In chapter 3, it is discussed that any reinforcement learning task is identified by the objects : agent, environment, states, actions, and rewards. To convert the problem of minimizing the navigation time into a reinforcement learning task, we identify these objects in the context of robot navigation as shown in Figure 7.3.

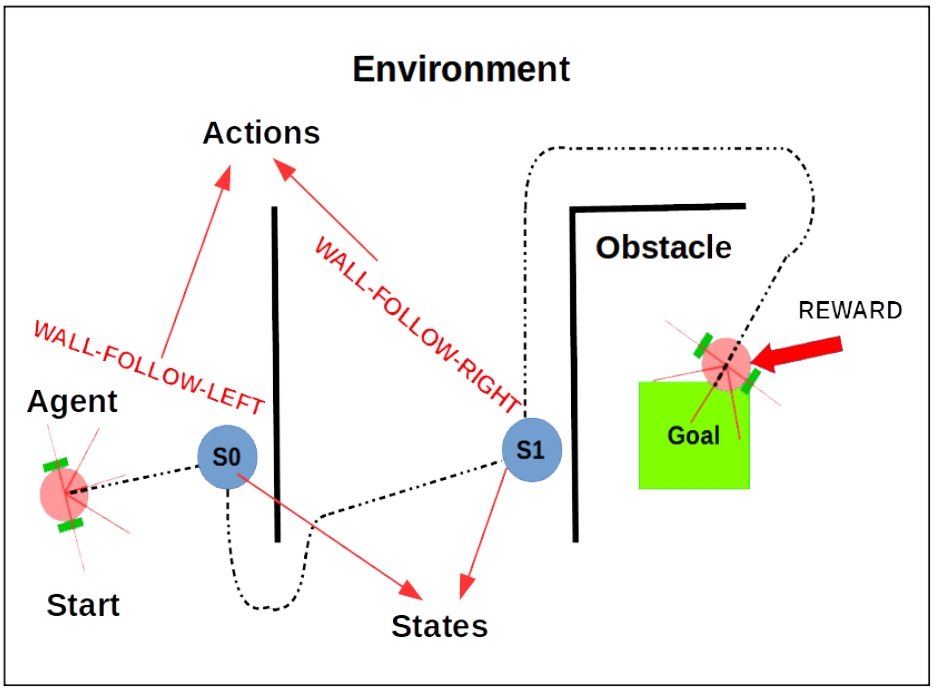

Figure 7.3: Reinforcement Learning Task Formulation for the Problem of Minimizing Navigation Time.

1. **Agent** : The mobile robot that we want to learn navigation to goal location in

minimum possible time is the agent.

2. **States** : The locations in the environment where the agent enters PRE-WALL-FOLLOW mode of navigation can be considered as the states for the reinforcement learning problem. In Figure 7.3, the states are shown as S_0 and S_1.

3. **Actions** : The agent can take select one of two actions in these states : WALL-FOLLOW-LEFT and WALL-FOLLOW-RIGHT.

4. **Rewards** : A reward is presented to the agent when it reaches the goal in such a way that the actions that result in shorter navigation time are strengthened more than the actions that result in longer navigation time.

Next we shall look at an important aspect of the reinforcement learning view of the problem.

7.3 Delayed Reward Nature of the Learning Problem

A simplified view of the reinforcement learning formulation of Figure 7.3 is shown in Figure 7.4.

We can observe that the reward is given to the agent only after reaching a goal which happens several time steps after the actions A_0 and A_1 have been taken. The robot must associate the reward with these actions that have been taken many timesteps in the past. This is the delayed reward problem introduced in chapter 3.

In order to solve this we use a spiking neural network to implement the solution to this reinforcement learning problem with delayed reward nature. Chapter 8 discusses the complete solution using eligibility trace introduced in chapter 5 to solve the delayed reward problem.

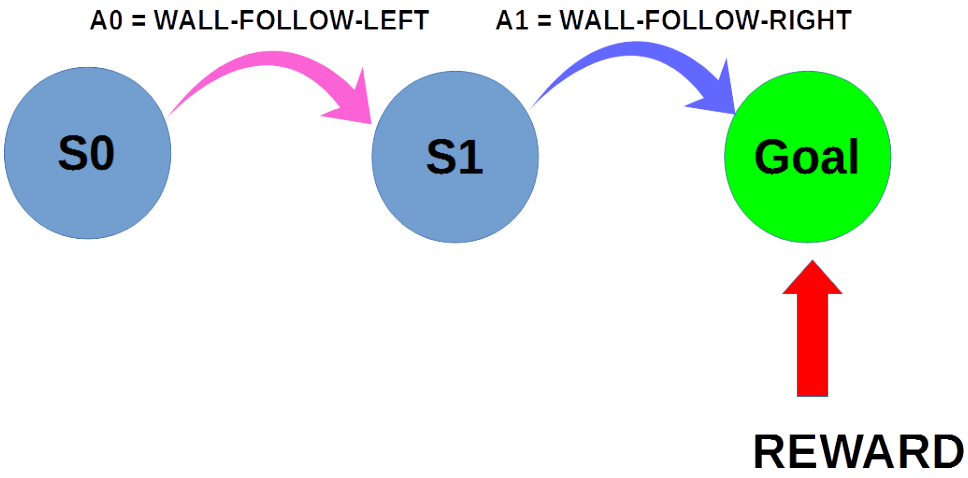

Figure 7.4: Simplified Reinforcement Learning Formulation.

8. THE COMPLETE SOLUTION *

Chapter 7 discusses a formulation of the navigation time minimization problem as a reinforcement learning task and argues that it is an inherently a delayed reward problem. This chapter presents a solution to the navigation problem, that is the outcome of this research. It is based on utilizing spiking neural networks discussed in chapter 4 to implement reinforcement learning as shown in chapter 5.

The proposed solution to the problem of navigating the mobile robot to a goal in shortest possible time is a spiking neural network architecture and its training algorithm. The role of this spiking neural network in solving the problem is depicted in Figure 8.1. The reward mechanism and action selection are explained in this chapter.

The spiking neural network is shown as a block in Figure 8.1. In chapter 7 we have seen how the wall following mode affects the navigation time. In this research a spiking neural network architecture is proposed that implements a reinforcement learning algorithm to select the wall following modes in such a way that the robot learns to navigate to the goal in shortest possible time.

In order to build the spiking neural network, we need the following

1. State representation.
2. State encoding for spiking neural network.
3. Action representation.
4. Synaptic connections to encode the value of (state, action) pair together with state and action representation.

*Parts of the material presented in this chapter are reprinted with permission from "Navigating mobile robots to target in near shortest time using reinforcement learning with spiking neural networks" by Amarnath Mahadevuni and Peng Li, 2017. *International Joint Conference on Neural Networks(IJCNN)*, pp.2243-2250, May 2017.

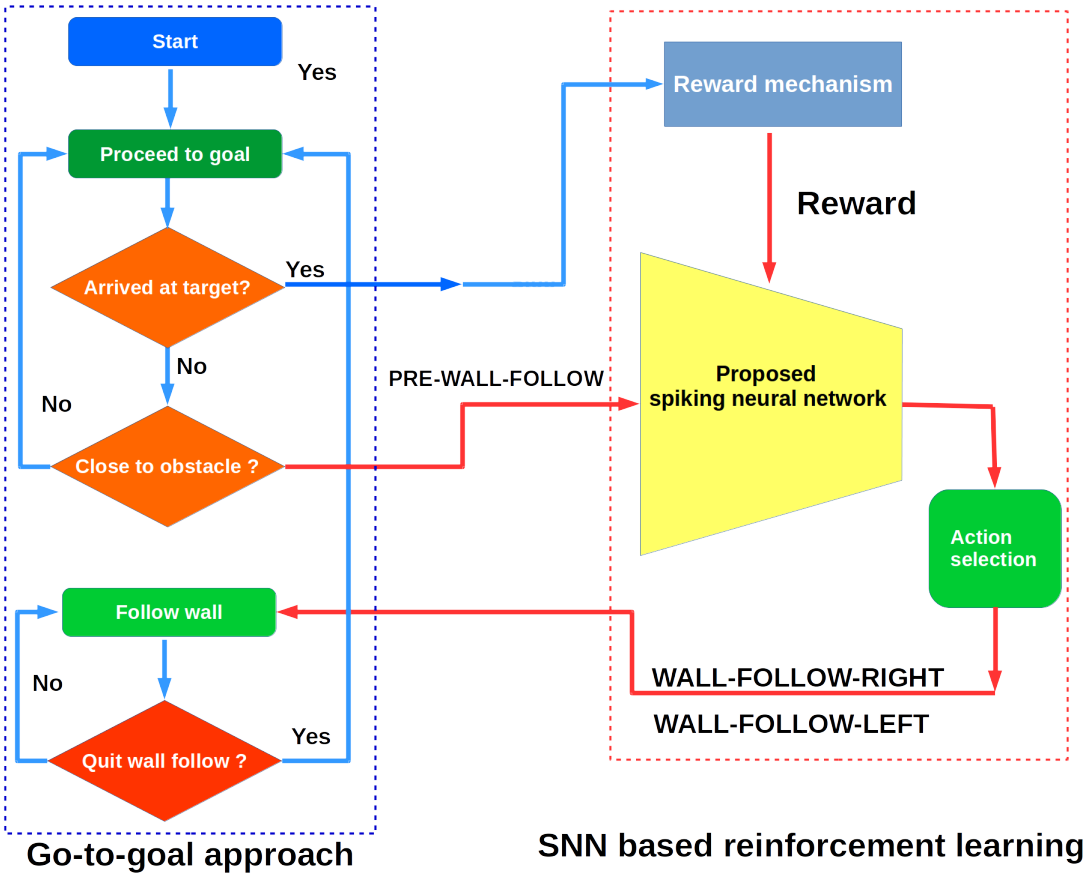

Figure 8.1: Proposed spiking neural network supplements the go-to-goal approach to minimize the navigation time.

5. Form a spiking neural network architecture using the state encoding, action encoding, and the synaptic connections.

Additionally, these aspects are needed to complete the reinforcement learning formulation.

1. Action selection.

2. Reward mechanism.

3. Handling the delayed reward problem.

We discuss each of these aspects next to build a spiking neural network and its learning algorithm.

8.1 State Representation

Chapter 7 identifies the states for the reinforcement learning formulation as the locations where the agent enters PRE-WALL-FOLLOW state. Figure 8.2 illustrates the robot in a PRE-WALL-FOLLOW navigation mode. The state is represented using two parameters: d and θ , that are calculated in the PRE-WALL-FOLLOW navigation mode of the trajectory.

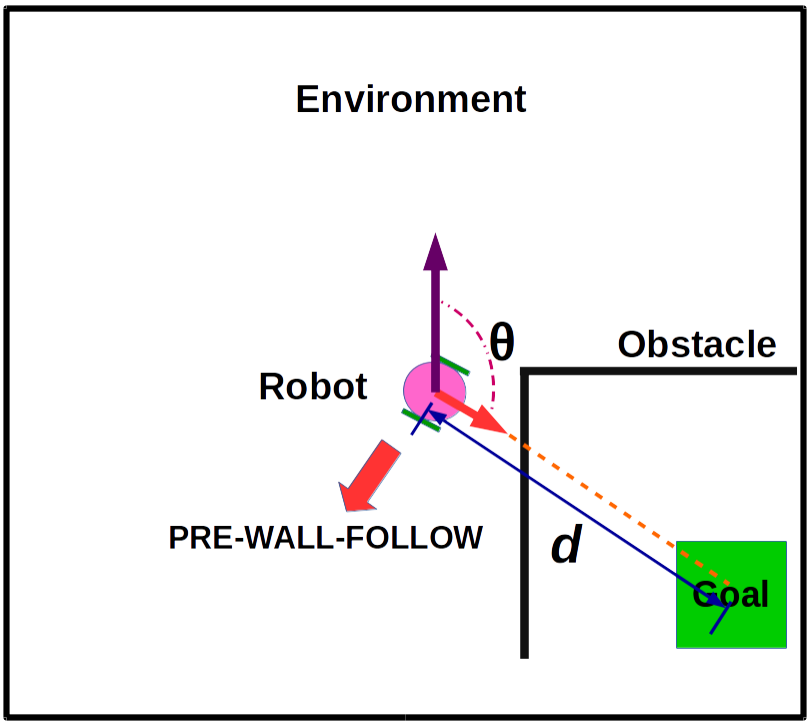

Figure 8.2: State representation. Adapted with permission from [1]. ©[2017] IEEE.

1. d : Straight line distance between the robot's center and the goal location. Its unit is pixel.

2. θ : Angle between the heading and a fixed vertical line in the 2-D environment. Its unit is degree.

Therefore in this representation a state is an ordered pair (θ, d). As such the state space is continuous.

8.2 State Encoding

To be able to use spiking neural networks to encode a state, a method to convert state representation to spike activity is needed. It is also necessary that the spiking activity of the population of neurons representing the state should differ from state to state. We use an approach similar to the state encoding scheme in [23], also used in [1]. The following definitions are needed to understand the encoding of states into spike activities.

1. Set $D = \{100, 200, 300, ..., 600, 700, 800\}$. This represents a discretized space for d values. Cardinality of D is 8.

2. Set $\Theta = \{0^0, 30^0, 60^0, ..., 300^0, 330^0\}$. This represents a discretized space for θ values. Cardinality of Θ is 12.

3. Set $C = D\ X\ \Theta = \{(100, 0^0), (100, 30^0), ..., (800, 330^0)\}$. This is the Cartesian product of sets D and Θ representing the discretized state space.

The cardinality of set C above is 96. Consider a layer of 96 spiking neurons each of which is assigned an ordered pair from set C . Therefore if a neuron is denoted by n_i where $i \in [0, 95]$, and an element of C is represented by C_i where $i \in [0, 95]$, then each neuron n_i maps to a point C_i. This mapping is shown in a rectangular box in Figure 8.3. Each circular object is a spiking neuron.

The encoding of a state represented by (d, θ) into spike activities is done by calculating the spike frequencies f_i for neuron n_i where $i \in [0, 95]$ as follows:

$$f_i = f_0 exp\left(-\frac{(d-d_i)^2}{2\sigma_1^2} - \frac{(\theta-\theta_i)^2}{2\sigma_2^2}\right) \tag{8.1}$$

Equation 8.1 ensures that the spiking activity of population of neurons representing a state should differ from one state to another.

f_0, σ_1 and σ_2 are design parameters. An example of spike activities is shown beside the state layer in Figure 8.3.

The state layer thus formed is a layer of spiking neurons that encodes state of the robot (agent). The computation of spike generation frequency using 8.1 produces an aliasing effect. This is discussed next.

8.2.1 Aliasing in State Representation

It is intended that the population of state neurons that spike with a considerable frequency ($> 20\ Hz$) at a state S_1 is as different as possible from the population of neurons producing spikes at a different state S_2. However there can be certain neurons that fire in both these states making the state representation not completely unique. This is due to the aliasing effect introduced by the Gaussian like computation of frequencies in 8.1. To analyze this, consider a state $S_1 = (d_1, \theta_1) = (150\ pixel, 70^0)$, and $S_2 = (d_2, \theta_2) = (300\ pixel, 120^0)$. Figure 8.4 shows the spike generation frequency on Y axis plotted calculated using 8.1 at the two states S_1 and S_2 for the state layer neurons represented on X-axis. The parameters of 8.1 used to plot Figure 8.4 are : $\sigma_1 = 60, \sigma_2 = 60, f_0 = 100Hz$. The circled region shows the overlap in the frequencies of populations of state layer neurons spiking at state S_1 and S_2. The key to reducing this aliasing effect is to change the parameters σ_1, σ_2. Figure shows a similar plot with these changed parameters:

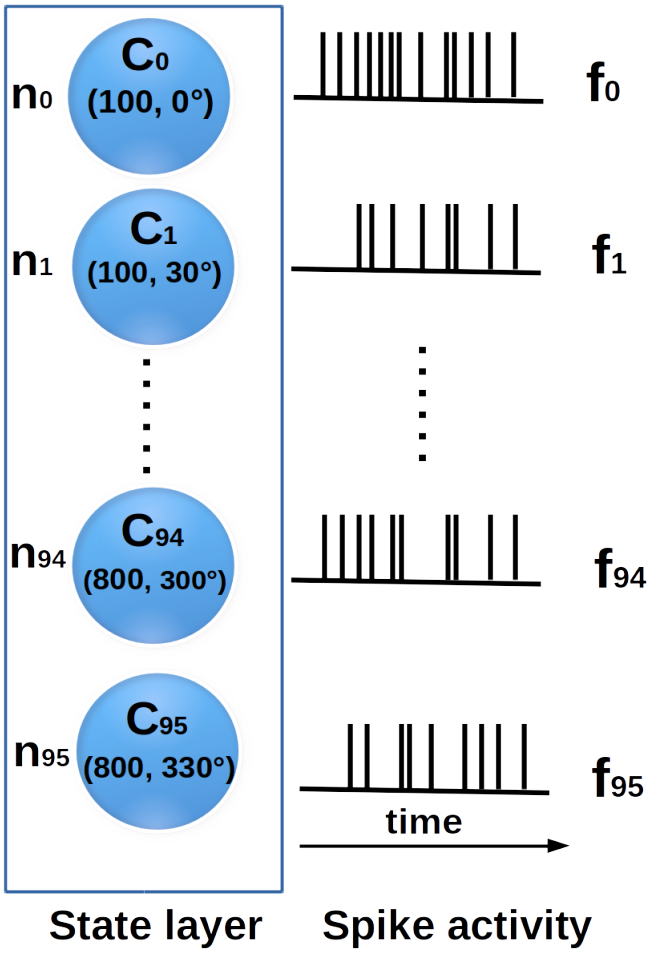

Figure 8.3: Stimulus layer and state encoding with spike activity with frequencies f_i for $i \in [0, 95]$

$\sigma_1 = 60, \sigma_2 = 10$. The overlap of state layer neuron populations spiking at states S_1 and S_2 is reduced with the new parameters.

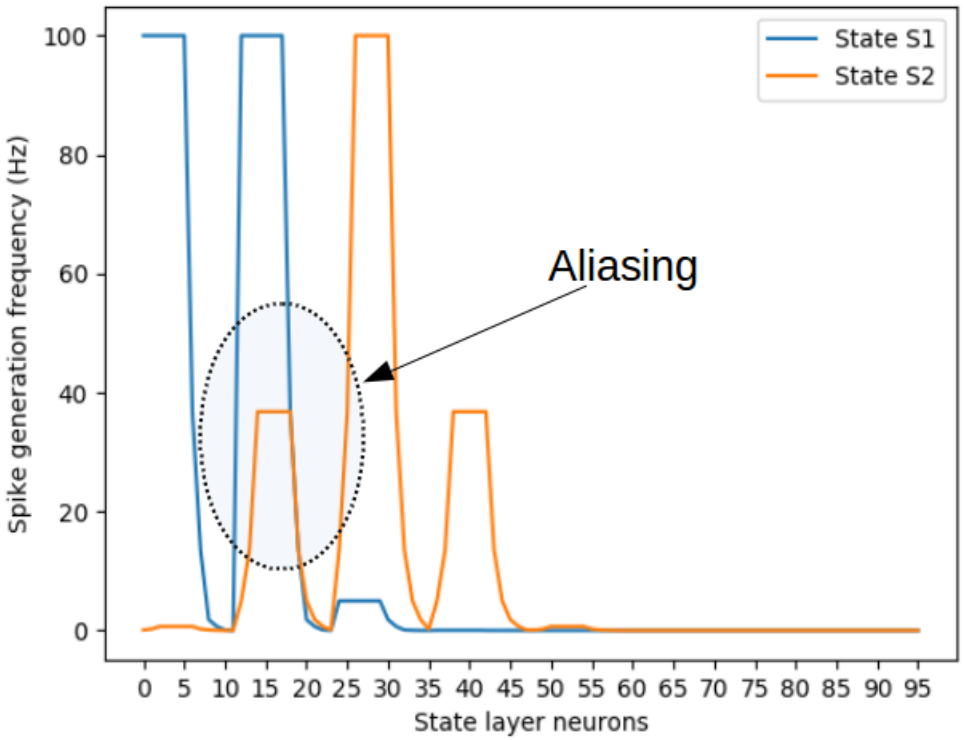

Figure 8.4: Spike generation frequency at states S_1 and S_2 using equation 8.1 with $\sigma_1 = 60, \sigma_2 = 60$.

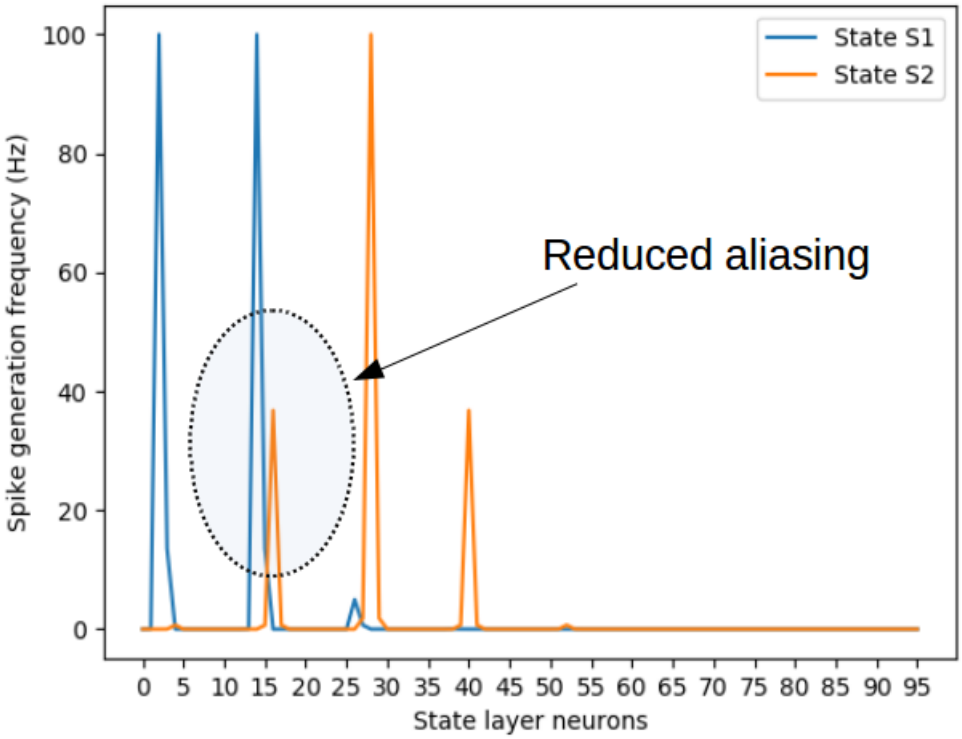

Figure 8.5: Spike generation frequency at states S_1 and S_2 using equation 8.1 with $\sigma_1 = 60, \sigma_2 = 10, f_0 = 100Hz$.

Thus the parameters for state layer neuron spike generation frequency in 8.1 are determined empirically. The effects of aliasing on learning performance is discussed further using an example in chpter 9.

8.3 Action Representation

At every state, the learning agent can take two actions : WALL-FOLLOW-LEFT and WALL-FOLLOW-RIGHT. Therefore two spiking neurons are chosen one of which represents WALL-FOLLOW-LEFT and other represents WALL-FOLLOW-RIGHT. These neurons are labeled LEFT and RIGHT respectively in Figure 8.6.

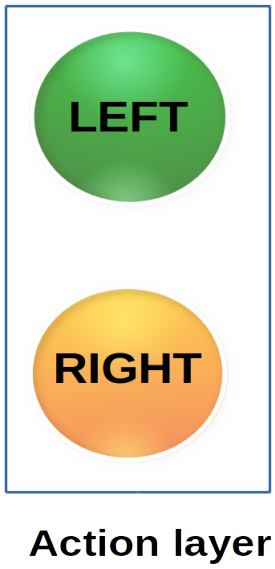

Figure 8.6: Action neurons

8.4 Encoding the Value of (State, Action) Pair

It is important to define a value function $V(s, a)$ for state s and action a for us to calculate importance of actions that can be taken in a given state. The state layer encodes the states and the action layer encodes actions, so it makes sense to connect them and define that the spike activity of neurons in action layer encodes the value function. This connectivity is shown in Figure 8.7.

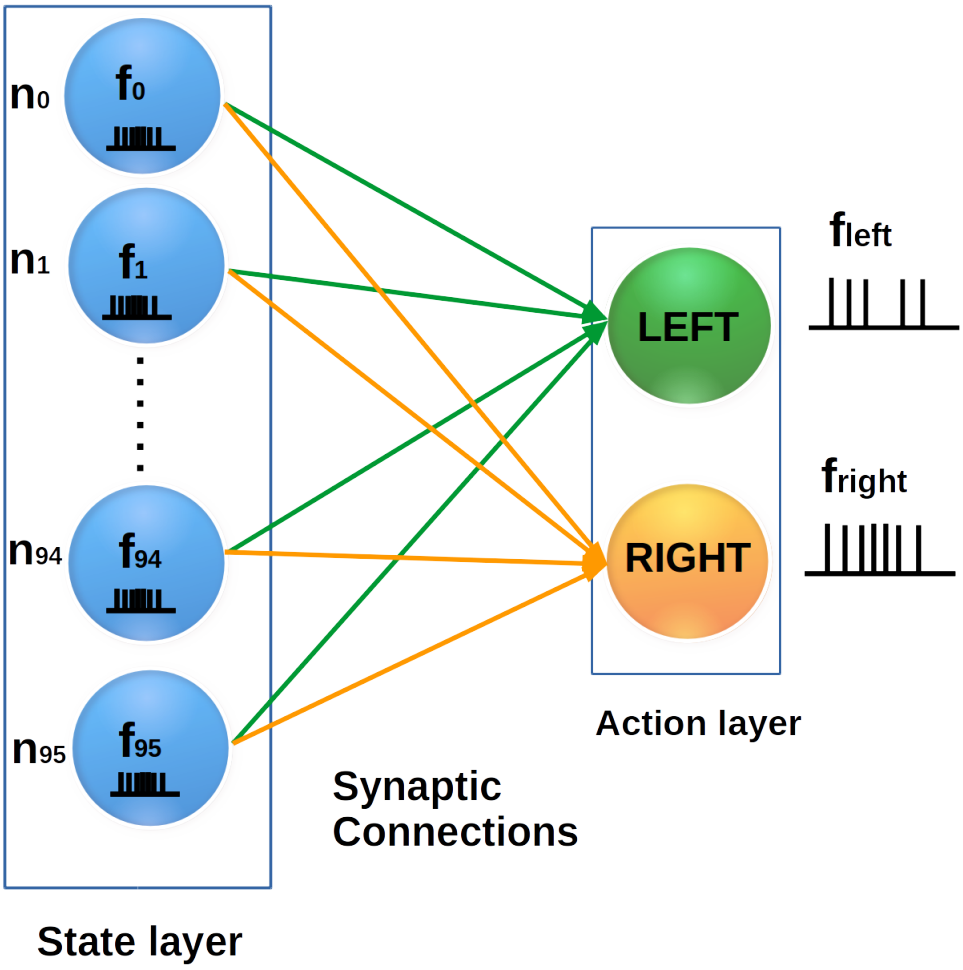

Figure 8.7: Spike activity of action layer neuron can be interpreted as value function for the state, and action represented by the neuron

Figure 8.7 shows the spike activity of action layer neurons represented by frequency labels f_i for $i \in [0, 95]$. For a state $s = (d, \theta)$, the state layer generates a spike activity shown in Figure 8.7, that results in a spike activity in action layer. The frequency of spikes for the neuron labeled "LEFT" is f_{left} and that for neuron labeled "RIGHT" is f_{right}. f_{left} can be interpreted as the value of state s and action WALL-FOLLOW-LEFT. Similarly

f_{right} can be interpreted as the value of state s and action WALL-FOLLOW-RIGHT.

8.5 The Spiking Neural Network Architecture

Figure 8.7 completes the building of our spiking neural network architecture. The synaptic weights support spike timing dependent plasticity as the primary way of learning. A simplified view is presented in Figure 8.8. The neurons are labeled by their respective spike generation frequencies f_i for $i \in [0, 95]$

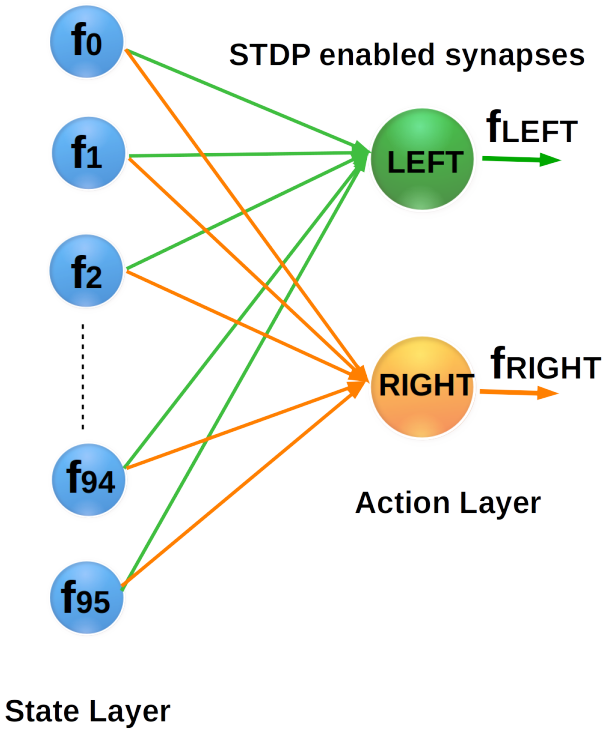

Figure 8.8: Simplified architecture of the spiking neural network. Adapted with permission from [1]. ©[2017] IEEE.

8.6 Integrating the Spiking Neural Network into the Navigation Algorithm

Figure 8.1 showed the spiking neural network as a computational block without the architecture. This section replaces this block with SNN (spiking neural network) of Figure 8.8. Figure shows the building blocks of the proposed spiking neural network based reinforcement learning algorithm integrated with the go-to-goal solution.

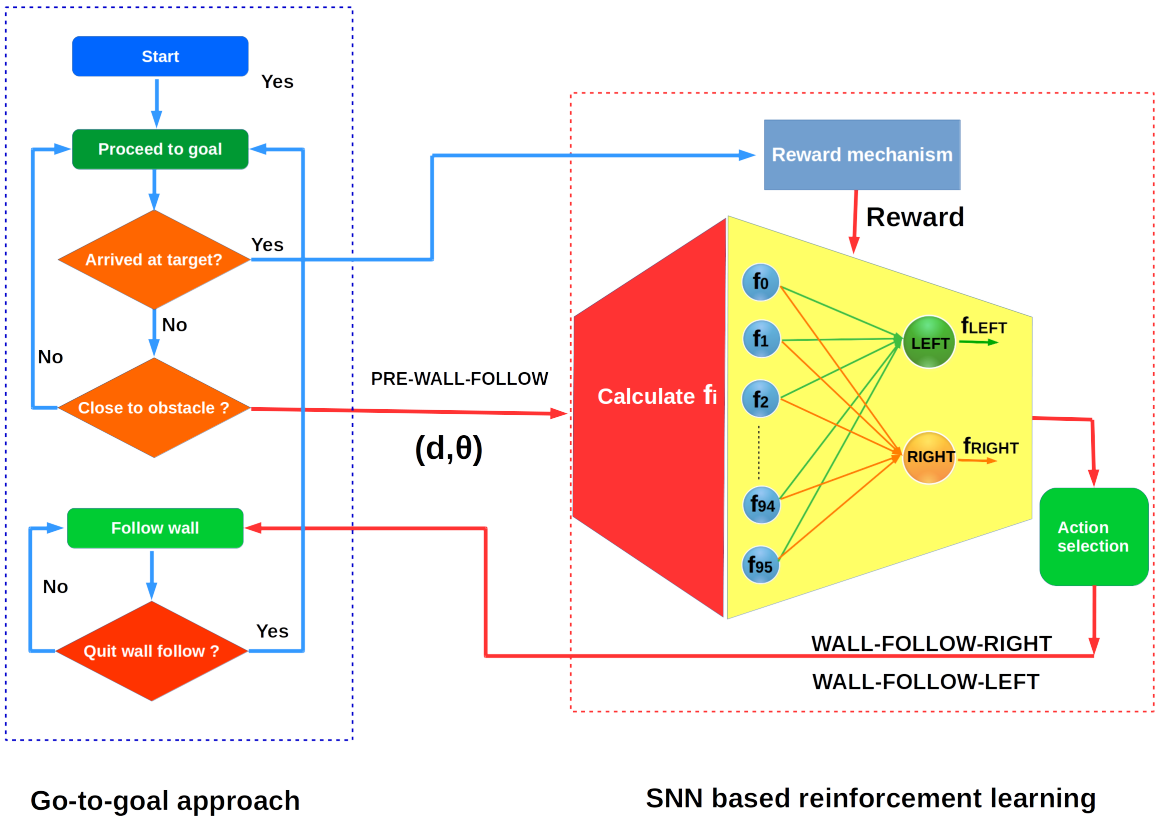

Figure 8.9: Integration of the designed SNN with the go-to-goal approach

The next section describes the process of training the network using action selection and reward mechanism.

8.7 Learning to Navigate in Shortest Possible Time

This section explains the proposed reinforcement learning algorithm in detail. A machine learning model learns over multiple trials. Hence the best way to understand the proposed algorithm is to understand a single trial. We fix the start position of the agent and location goal constant throughout the learning process. In each trial the agent navigates from start to goal locations using the go-to-goal approach described in chapter 6. Depending on the location of obstacles, the agent may encounter multiple PRE-WALL-FOLLOW states. In the Figure 8.10, the agent encounters two PRE-WALL-FOLLOW states labeled S_0 and S_1.

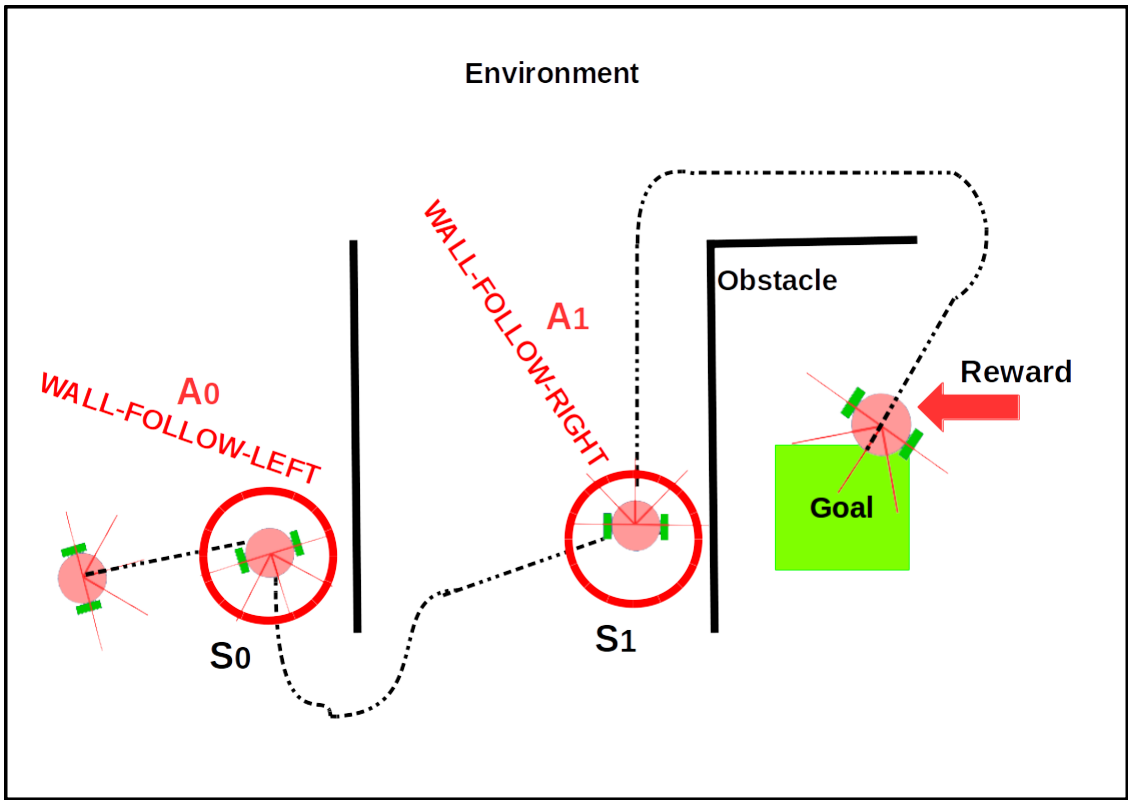

Figure 8.10: Integration of the designed SNN with the go-to-goal approach

Learning in the robot in a trial happens by training the spiking neural network weights. This involves three components : Action selection, activating eligibility traces for synaptic weights of SNN, reward modulation of synaptic weight changes of SNN. These are shown in Figure 8.11 along with the go-to-goal navigation states where they take place.

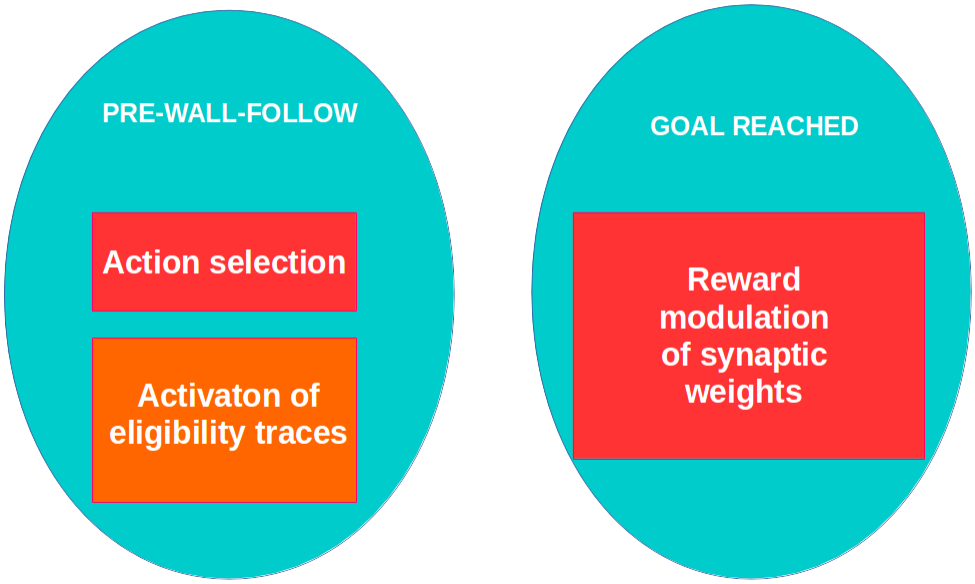

Figure 8.11: Components of the Spiking Neural Network Training

These three components of learning are described next.

8.7.1 Action Selection

Action selection is done by the robot in PRE-WALL-FOLLOW navigation state. It selects an action from the set of wall following behaviors WALL-FOLLOW-LEFT, WALL-FOLLOW-RIGHT. The steps involved in action selection are :

1. Get the state representation of the robot : (d, θ), calculated as shown in Figure 8.2.

2. Calculate the frequencies f_i using equation 8.1 for $i \in [0, 95]$.

3. Run the SNN of Figure 8.8 for a duration of $T\ ms$ by applying the spike generation frequencies f_i to the corresponding neurons n_i for $i \in [0, 95]$. This results in spike activity in the action layer of the SNN. Neuron "LEFT" outputs a spike train with frequency f_{left} and neuron "RIGHT" outputs a spike train with frequency f_{right}.

4. Select a wall following behavior from the set of actions WALL-FOLLOW-LEFT, WALL-FOLLOW-RIGHT using exploration-exploitation process.

   (a) **Exploration** : Select an action WALL-FOLLOW-LEFT or WALL-FOLLOW-RIGHT with 50% probability.

   (b) **Exploitation** : Select action WALL-FOLLOW-LEFT if $f_{left} \geq f_{right}$. Select action WALL-FOLLOW-RIGHT otherwise.

5. Exploitation can happen with a probability ϵ where $0 < \epsilon < 1$ and exploration happens with probability $1 - \epsilon$.

8.7.2 Activating Eligibility Traces

When the spiking neural network is run in the action selection step, synaptic weight changes are calculated using STDP equation 4.7. To understand the activation of eligibility traces, the concept of a winning action layer neuron is essential. If the action selection step selects WALL-FOLLOW-LEFT, then the winning neuron is "LEFT", otherwise the winning neuron is "RIGHT". Eligibility trace is then activated as follows:

1. Discard the STDP weight changes of the synapses connecting the state layer to the non-winning action layer neuron. For example, if the agent selects WALL-FOLLOW-LEFT action, discard the STDP weight changes accumulated for synapses incoming to the "RIGHT" action neuron.

2. Decay the STDP weight changes every time step by a decay parameter β until the robot reaches the GOAL-REACHED state using equation 5.1.

8.7.3 Reward Modulation of Synaptic Weight Changes

This component happens in GOAL-REACHED navigation state when the robot has arrived at the goal. The wall following decisions taken by the robot are rewarded in such a way that at a PRE-WALL-FOLLOW state on its trajectory, the action that takes the robot to the goal in shorter time is rewarded higher than the action that takes the robot to the goal in longer time. This discrimination in presenting the rewards must be encoded in the learning algorithm. Coincidentally, the eligibility trace measure described in 8.7.2 captures this difference in the goodness of an action.

For a given PRE-WALL-FOLLOW state, the eligibility traces of synapses incoming to the winning neuron are inversely proportional to the time taken by the robot to reach the target using that winning action. Hence eligibility trace directly measures the goodness of an action, along with memorizing the action taken to solve the delayed reward problem. Therefore eligibility trace in the proposed spiking neural network training has two purposes :

1. To memorize the actions taken during a trial, so that they can be rewarded even after a delay of several time steps.

2. Across multiple trials, it serves as a discriminator between longer time taking actions and shorter time taking actions.

Now, synaptic weight changes are applied to all synapses with activated eligibility traces as follows :

$$W_{ij} = W_{ij} + \Delta W_{ij}(t_0) * \beta^m * reward * \delta(t - t_{reward}) \tag{8.2}$$

where W_{ij} is the synaptic weight connecting a state layer neuron i to action layer

neuron j. t_0 is the time step at which the robot reaches a PRE-WALL-FOLLOW point on its trajectory. At t_0, the STDP weight change ΔW_{ij} is calculated using equation 4.7 by running the network for duration $T\ ms$ as discussed in section 8.7.1. $\delta(t - t_{reward}) = 1$ when the reward is applied. m is the number of time steps taken by the robot after t_0 to reach the goal.

This is same as equation 5.3. The $reward$ parameter for this specific research is chosen to be a constant, as the eligibility trace sufficiently discriminates good actions from bad actions. It this were not the case, then the reward function will have to be carefully designed to bring out the desired behavior.

8.8 Criteria to Stop Learning

Learning is stopped when the following conditions are satisfied:

1. At a given state, if $|f_{left} - f_{right}| > f_{threshold}$, where $f_{threshold}$ is a design parameter, the actions are selected using exploitation, and eligibility traces are not activated.

2. If $|f_{left} - f_{right}| > f_{threshold}$ for all PRE-WALL-FOLLOW states the robot encounters, then the learning is said to be complete, and the synaptic weights are fixed.

The learning algorithm can be best understood by looking at a simulated navigation example.

8.9 Understanding the Learning Algorithm with an Example

Consider an example environment "Env1" in Figure 8.12. This is a simulated example. The simulation details are presented next. This section presents a qualitative explanation of the learning algorithm.

To understand the learning algorithm, consider the eligibility traces of actions the robot takes at PRE-WALL-FOLLOW state labeled 1 in Figure 8.13 and 8.14.

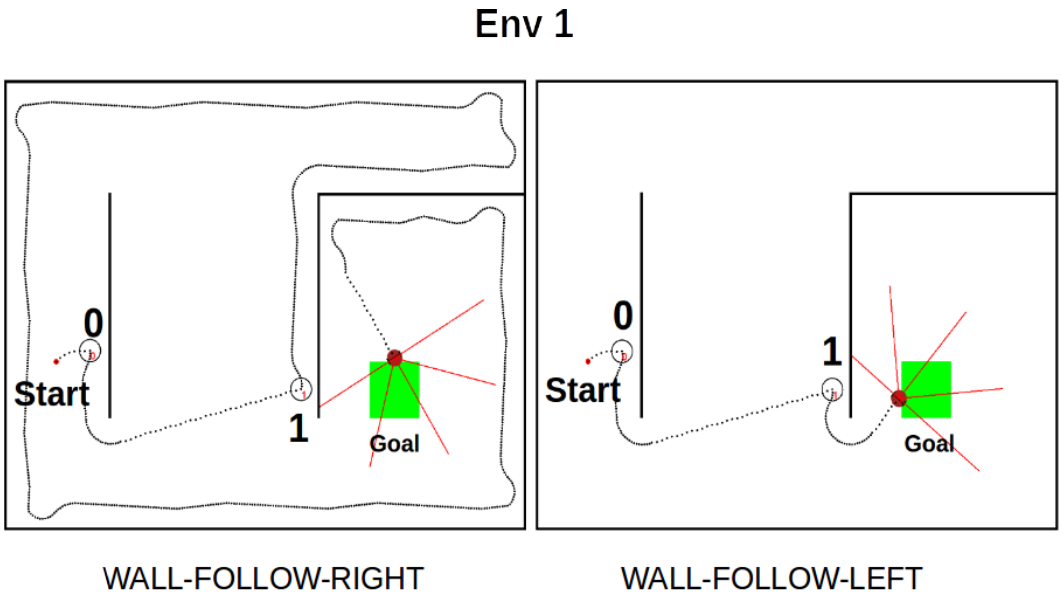

Figure 8.12: Trials selecting WALL-FOLLOW-RIGHT and WALL-FOLLOW-LEFT in Env 1 at state 1. Adapted with permission from [1]. ©[2017] IEEE.

Figure 8.13 shows eligibility trace for synaptic weights incoming to the neuron "RIGHT". The timestep at which eligibility traces are activated is marked by a vertical line labeled "1". The timestep when the robot arrives at the goal is indicated by label "Goal". We can see that by this timestep, the eligibility traces have negligible values. This is because, the decision WALL-FOLLOW-RIGHT taken at state 1 in the left subfigure of Figure 8.12 takes the robot to goal in a long time. On the other hand, Figure 8.14 trace shows eligibility trace for synaptic weights incoming to the neuron "LEFT" neuron. The timestep at which eligibility traces are activated at state 1 is marked by a vertical line labeled "1". The timestep when the robot arrives at the goal is indicated by label "Goal". We can see that by this timestep, the eligibility traces have significantly larger values than that for action WALL-FOLLOW-RIGHT. Therefore the synaptic weights incoming to neuron "LEFT" are strengthened more than the synaptic weights incoming to neuron "RIGHT". After learning is converged, the agent favors the action WALL-FOLLOW-LEFT more than

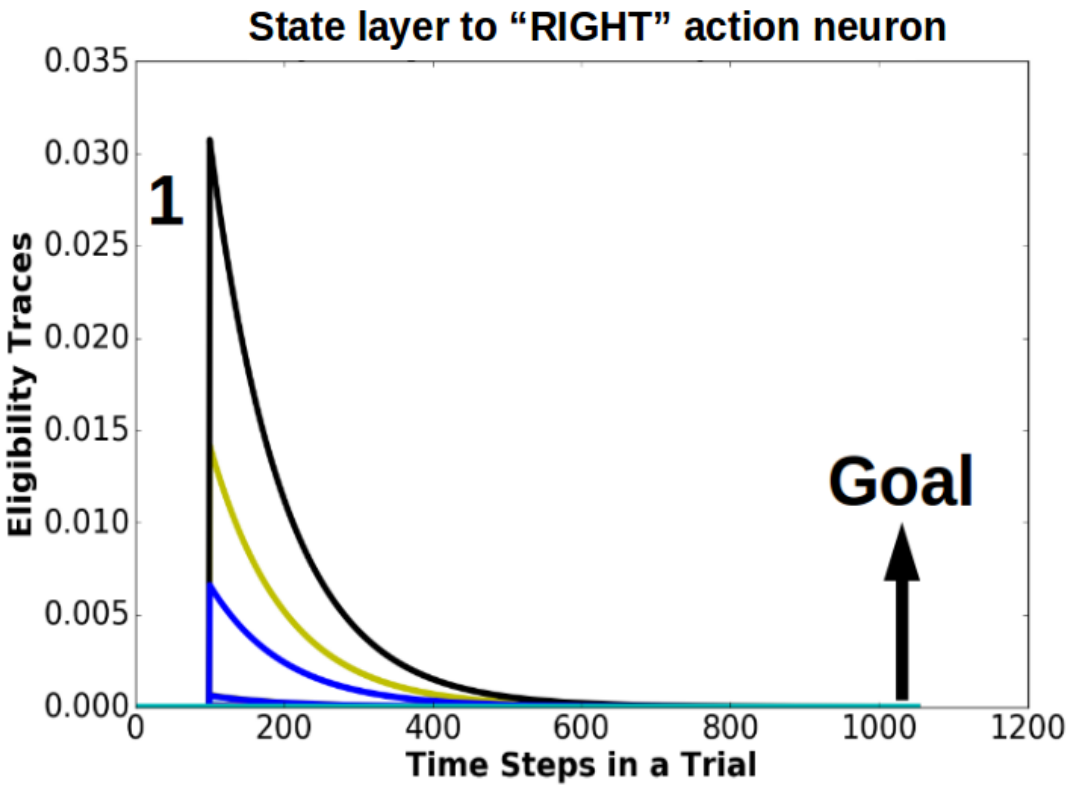

Figure 8.13: Eligibility traces of synapses incoming to winning neuron "RIGHT" in Env1. Adapted with permission from [1]. ©[2017] IEEE.

WALL-FOLLOW-RIGHT at state 1.

Next chapter discusses simulation of the learning algorithm, experimental setup, and results.

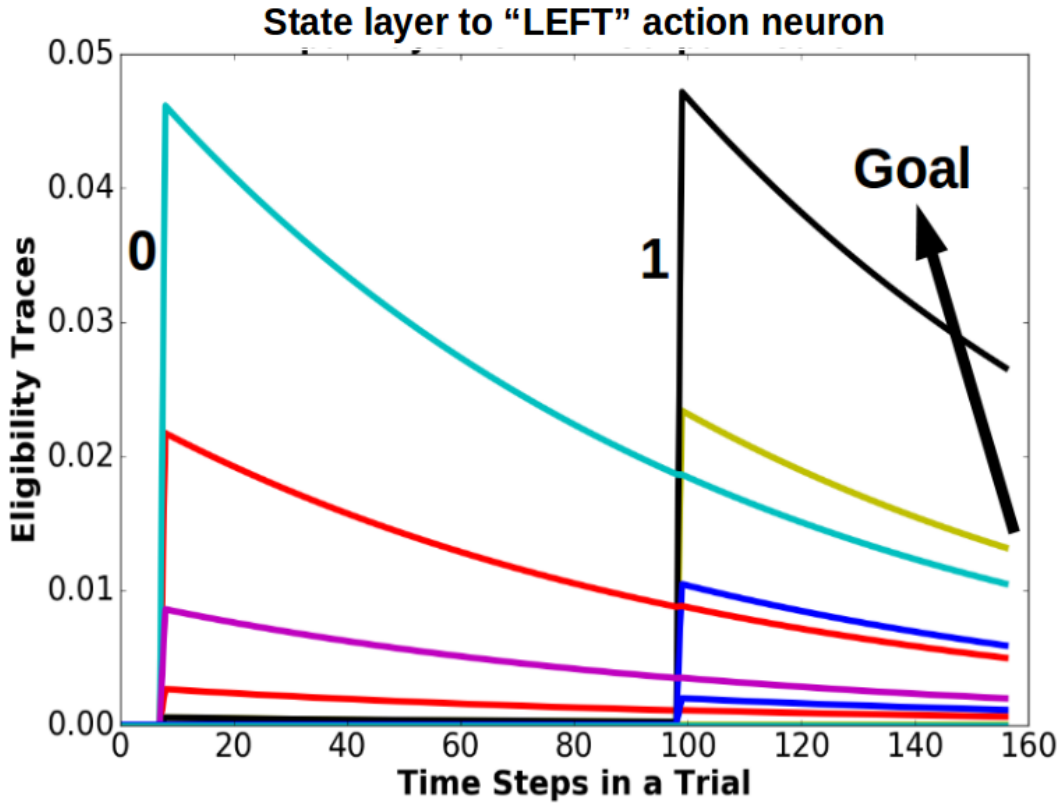

Figure 8.14: Eligibility traces of synapses incoming to winning neuron "LEFT" in Env1. Adapted with permission from [1]. ©[2017] IEEE.

9. RESULTS AND CONCLUSION *

9.1 Experimental Setup

The experimental setup for this research consists of the following components :

1. **2-D navigation environment** : A graphics 2-D environment is created using a C++ game library allegro (http://liballeg.org). An example is shown in Figure 9.1. The obstacle and goal configuration can be changed using C++. The dimensions chosen for the 2-D environment are 1000 pixels X 800 pixels. The obstacles are wall-like rectangular shaped objects. The goal location is labeled "Goal".

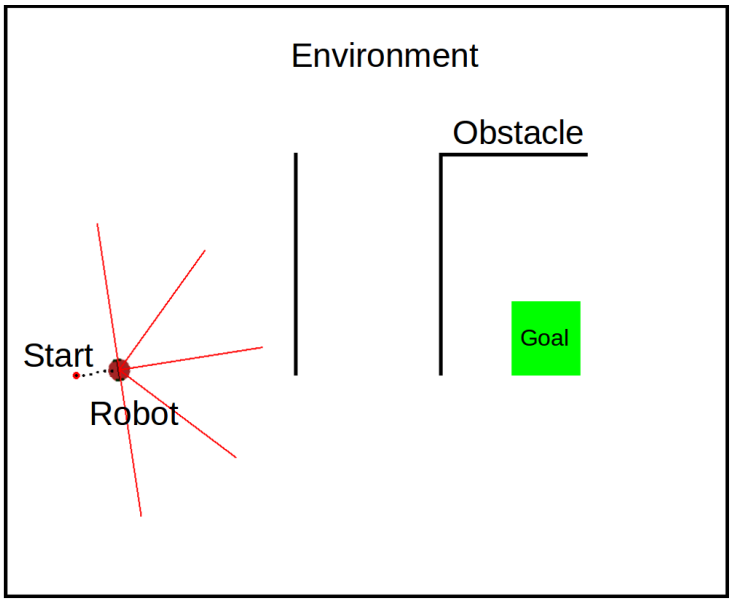

Figure 9.1: An Example of 2-D navigation environment. Reprinted with permission from [1]. ©[2017] IEEE.

*Parts of the material presented in this chapter are reprinted with permission from "Navigating mobile robots to target in near shortest time using reinforcement learning with spiking neural networks" by Amarnath Mahadevuni and Peng Li, 2017. *International Joint Conference on Neural Networks(IJCNN)*, pp.2243-2250, May 2017. ©[2017] by IEEE.

2. **Robot abstraction** : A mobile robot abstraction (labeled "Robot" in Figure 9.1) of Figure 9.2 is realized using a C++ object that can move in the 2-D navigation environment. The robot navigation dynamics described in chapter 2 is also written using allegro.

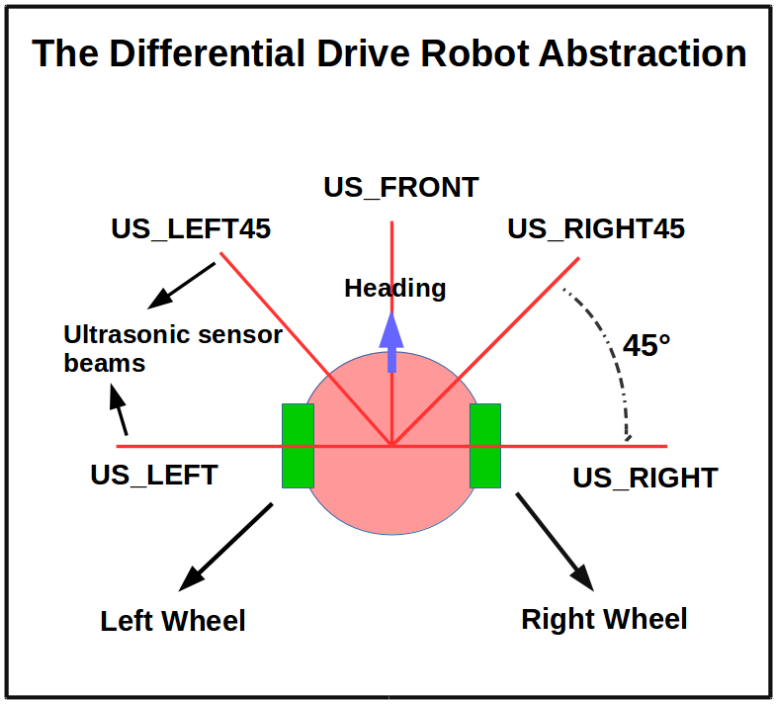

Figure 9.2: Differential drive robot abstraction. Reprinted with permission from [1]. ©[2017] IEEE.

3. **Spiking neural network simulator** : A C++ based spiking neural network simulator CARLsim [24] is used to simulate the neural network built in chapter 8. CARLsim uses Izhikevich spiking neuron model [18] [24]. For the synapses, CARLsim supports nearest neighbor STDP discussed in chapter 4. Eligibility traces are implemented as decayed STDP weight changes exactly as discussed in chapter 5.

9.2 Simulation

Simulation of the proposed robot navigation learning algorithm has two parts that interact with each other as shown in Figure 9.3

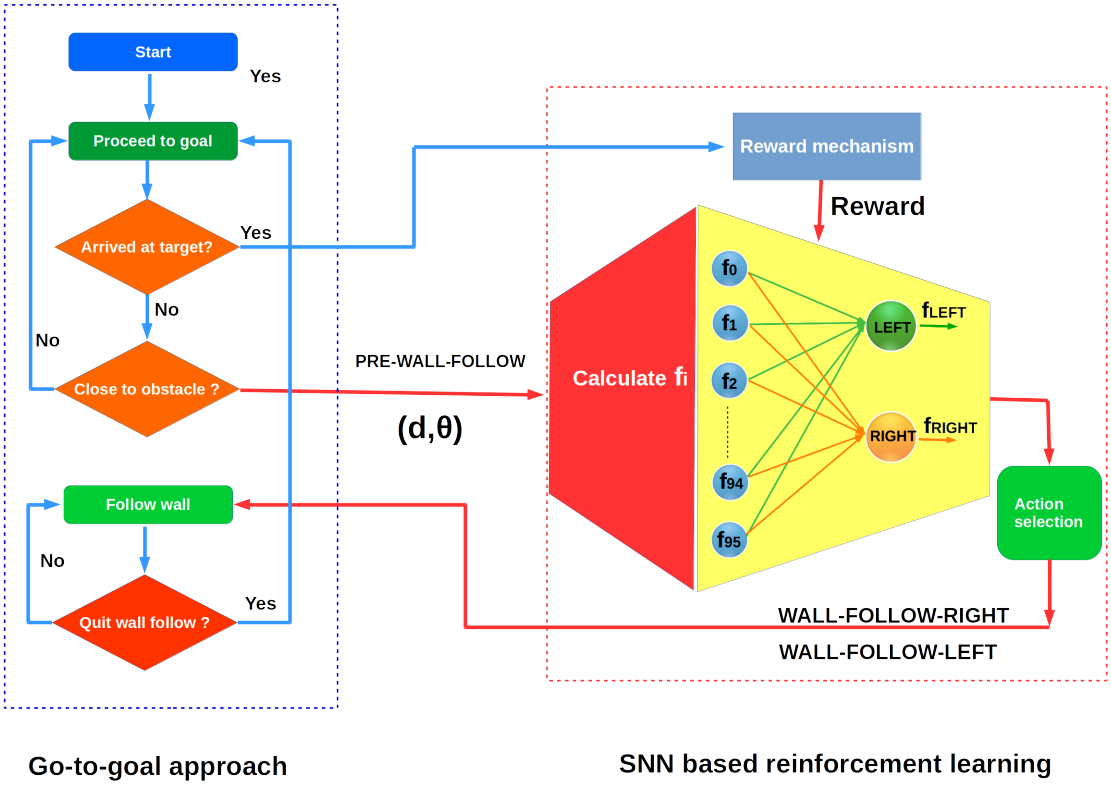

Figure 9.3: Interaction between go-to-goal approach and SNN based reinforcement learning approach.

1. **Go-to-goal navigation** : Written as a state machine in C++ utilizing the same states as described in chapter 6. The states are shown again in Figure 9.4. The Movement of graphic object implementing the robot is simulated using allegro.

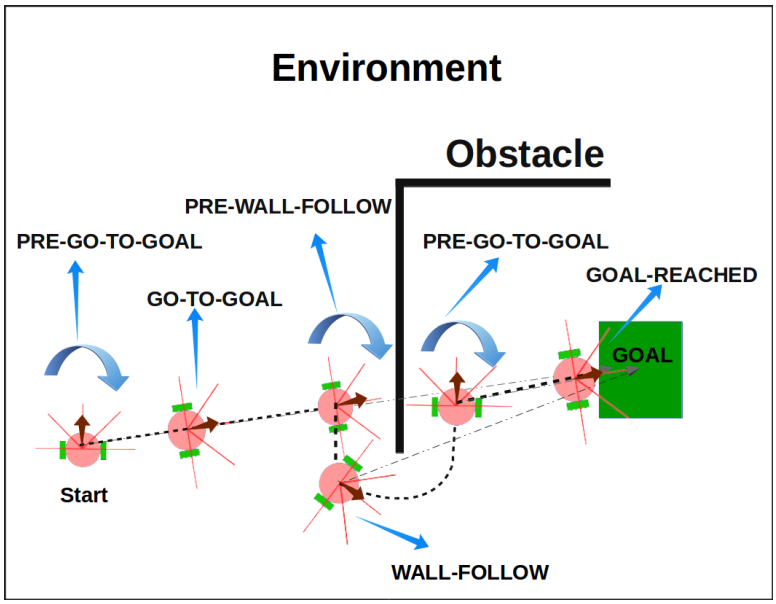

Figure 9.4: Navigation states of the robot trajectory in go-to-goal solution. Reprinted with permission from [1]. ©[2017] IEEE.

2. **Spiking neural network based learning** : In PRE-WALL-FOLLOW state of robot navigation, the spiking neural network proposed in chapter 8 is run using CARLsim to generate action layer neurons spiking frequencies. The neural network is shown again in Figure 9.3 in conjunction with go-to-goal navigation. The process of action selection, eligibility trace activation, and rewarding mechanism is described in detail in 8.7.

9.3 Results

The parameter values for various equations used in the simulation are shown in table 9.1.

Simulation of the learning algorithm is performed on three environments : Env1, Env2, and Env3. They differ in the configuration of obstacles and goal. The navigation trajectories before and after training are shown in Figures 9.5, 9.6, and 9.7 for Env1, Env2,and Env3 respectively.

Table 9.1: Simulation parameters and their values. Adapted with permission from [1]. ©[2017] IEEE.

Parameter	Description	Value
A_+	CARLsim LTP parameter for STDP in eq 4.5	0.0001
A_-	CARLsim LTD parameter for STDP in eq 4.6	0.0005
τ_+	CARLsim LTP parameter for STDP in eq 4.5 1	20ms
τ_-	CARLsim LTD parameter for STDP in eq 4.6	20ms
β	Eligibility trace decay in eq 5.1	0.99
f_0	Constant in eq 8.1	100 Hz
σ_1	Constant in eq 8.1	25 pixels
σ_2	Constant in eq 8.1	10°
$reward$	Scalar reward on reaching the Goal	10
$f_{threshold}$	Threshold firing rate in stopping criteria 8.8	$50Hz$

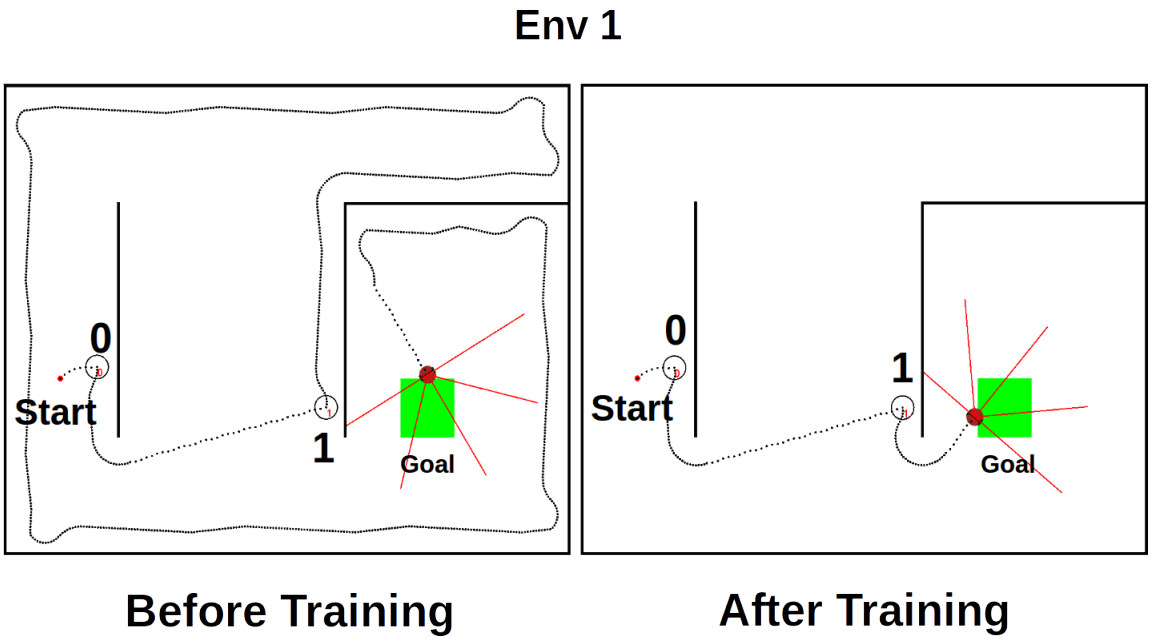

Figure 9.5: Simulation results for Env 1. Reprinted with permission from [1]. ©[2017] IEEE.

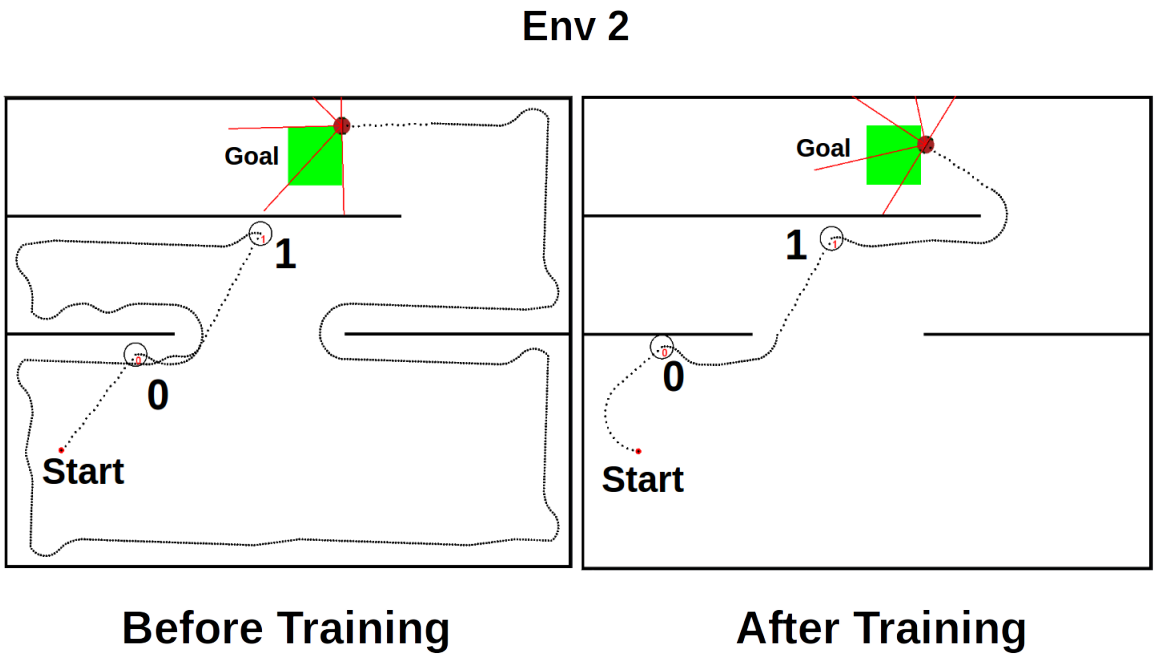

Figure 9.6: Simulation results for Env 2. Reprinted with permission from [1]. ©[2017] IEEE.

Table 9.2: Number of Trials taken for the learning to converge. Reprinted with permission from [1]. ©[2017] IEEE.

Environment	No. of Trials
Env1	14
Env2	16
Env3	44

In all three environments, the robot learns to select appropriate actions in PRE-WALL-FOLLOW states to reach the goal in shortest possible time within go-to-goal solution.

The number of trials taken for learning to converge is shown in table 9.2. Env3 takes more trials to converge as it has more complex configuration of obstacles than Env1 and Env2.

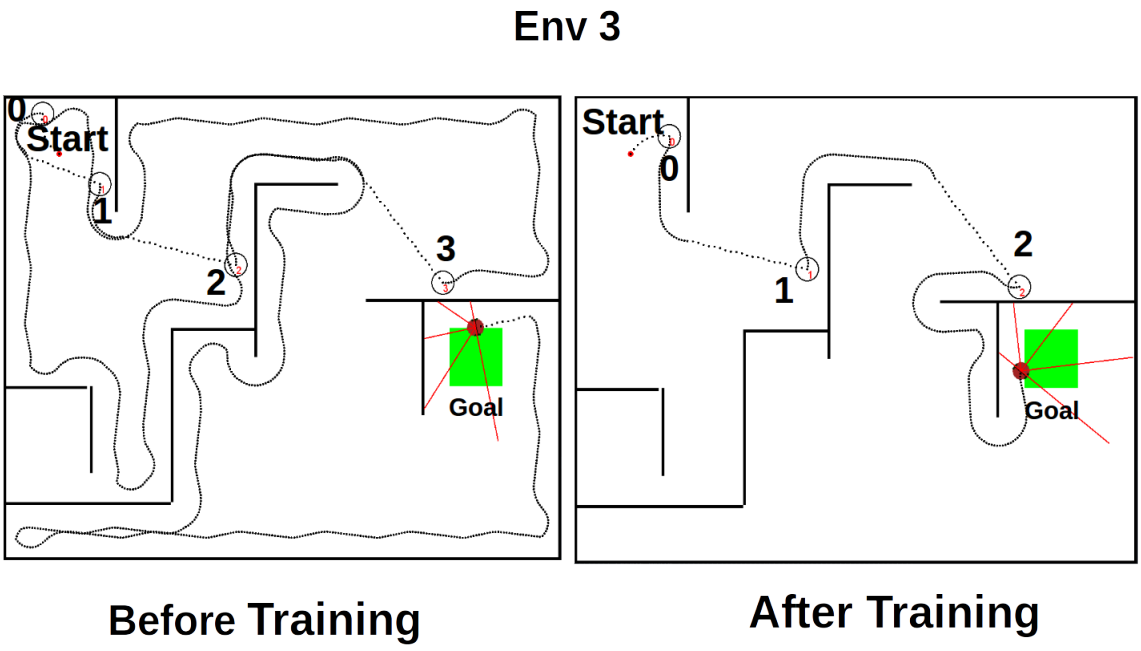

Figure 9.7: Simulation results for Env 3. Reprinted with permission from [1]. ©[2017] IEEE.

Table 9.3: Number of time steps taken by robot to each goal

Environment	Go-to-goal	Go-to-goal with SNN based learning
Env1	1021	222
Env2	604	165
Env3	1769	462

9.3.1 Comparison with Results without the Learning Algorithm

From table 9.3, we can see a significant reduction in number of navigation time steps to reach a goal location when SNN based learning is used with go-to-goal approach when compared with just the go-to-goal approach.

9.3.2 Impact of Aliasing in State Representation on Learning Performance

Section 8.2.1 of chapter 8 discusses the aliasing effect in the spike frequency generation of state layer of the spiking neural network when spike frequencies are calculated using equation 8.1. This section discusses the effect this aliasing can have on learning performance.

For the sake of comparing the algorithm with aliasing and no-aliasing, consider a simplified definition of performance of learning algorithm :

Performance : Number of trials taken by learning algorithm to converge.

The effect of aliasing is explained with a navigation example. Consider two trials of the learning algorithm shown in Figures 9.8 and 9.9.

The two states of interest are marked as S_1 and S_2. Their representation is $S_1 = (227.93\ pixel, 132.07^0)$ and $S_2 = (198.19\ pixel, 63.03^0)$. Now consider two cases:

9.3.2.1 With Aliasing

Parameters σ_1, σ_2 of 8.1 are chosen such that the state layer spike generation frequencies at states S_1 and S_2 exhibit aliasing. This is shown in Figure 9.10 plotted using $\sigma_1 = 60\ pixel, \sigma_2 = 60^0$.

The spike generating populations of state layers at states S_1 and S_2 overlap with each other significantly.

9.3.2.2 Without aliasing

Parameters σ_1, σ_2 of 8.1 are chosen such that the state layer spike generation frequencies at states S_1 and S_2 do not exhibit such extreme aliasing. This is shown in Figure 9.11 plotted using $\sigma_1 = 60\ pixel, \sigma_2 = 10^0$.

The spike generating populations of state layers at states S_1 and S_2 do not overlap with each other significantly. Next a comparison of performance using these two schemes is

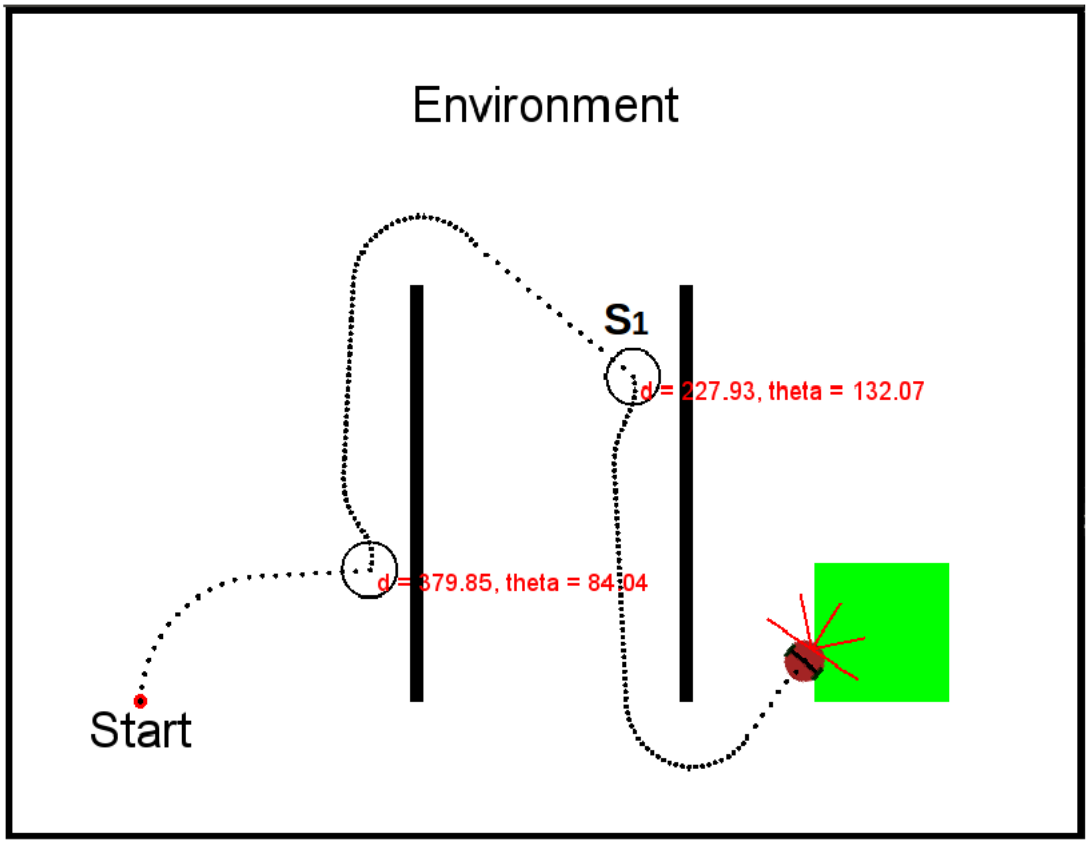

Figure 9.8: A trial showing state S_1.

shown.

9.3.2.3 Comparison

As shown in table 9.4, the number of trials taken by the robot to learn actions for shortest time path is only 17 with no aliasing scheme while it takes 42 trials to learn the optimal actions with aliasing.

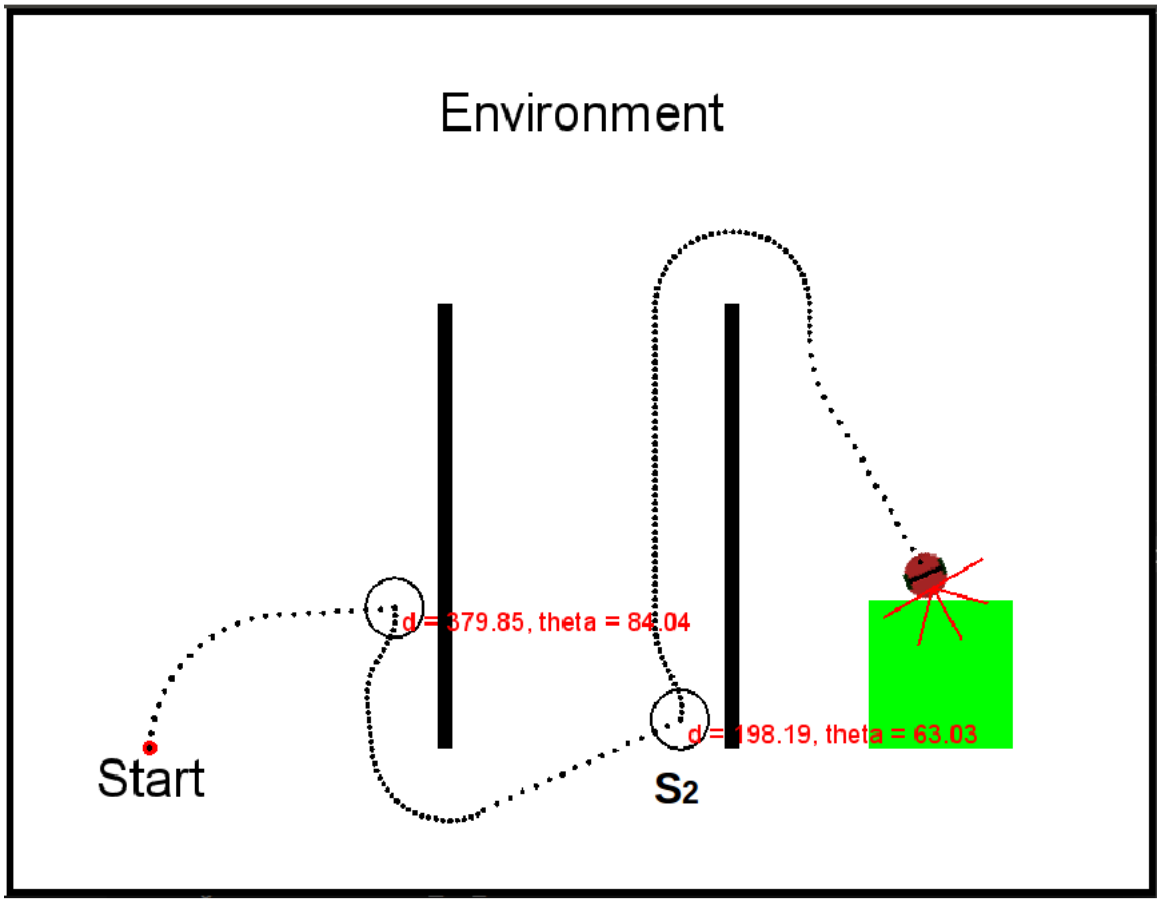

Figure 9.9: A trial showing state S_2.

Table 9.4: Performance : Number of trials to converge

With aliasing	Without aliasing
42	17

The parameters σ_1 and σ_2,shown in table 9.1, are selected empirically for simulation to calculate state layer frequencies such that the effect of aliasing is minimal.

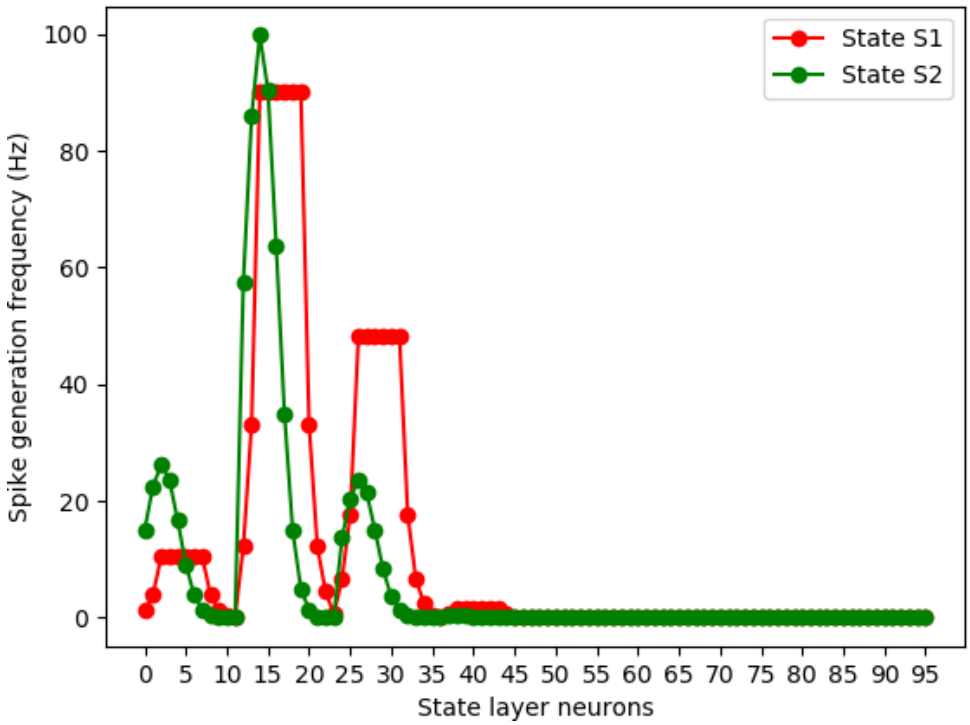

Figure 9.10: Aliasing when $\sigma_1 = 60\ pixel, \sigma_2 = 60^0$.

Next we discuss why aliasing affects performance.

9.3.3 Discussion

With aliasing, the state encoding of two states S_1 and S_2 is almost identical. At $S1$, the action WALL-FOLLOW-RIGHT is favorable while at S_2, the action WALL-FOLLOW-LEFT is favorable. These two actions are almost equally reinforced by the learning algorithm because the encodings of S_1 and S_2 are close to each other. Therefore, it takes larger number of trials for the frequencies of winning action neuron and losing action neuron to differ by $f_{threshold}$. (section 8.8). Thus aliasing increases the number of trials to converge, and decreases the performance of the algorithm.

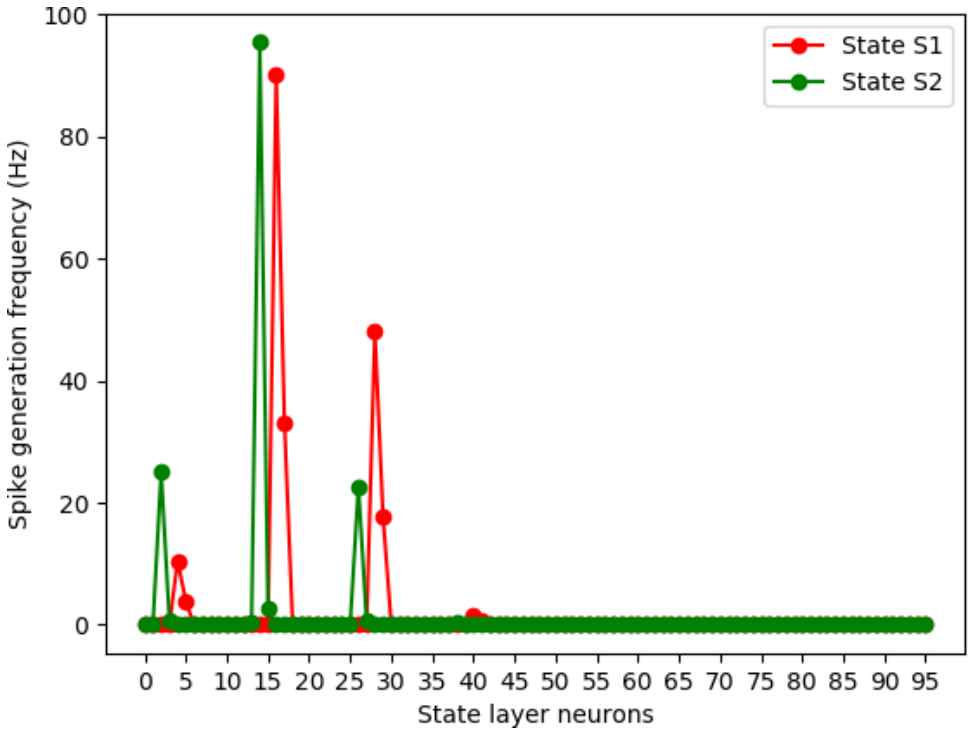

Figure 9.11: Elimination of aliasing when $\sigma_1 = 60\ pixel, \sigma_2 = 10^0$.

9.4 Conclusion

The research goal of minimizing the navigation time from a start location to goal location is realized using a spiking neural network implementation of reinforcement learning algorithm. The following are some advantages and disadvantages of this approach :

Advantages

1. Easy to understand and implement.
2. Computationally efficient because :
   (a) The state space consists only of PRE-WALL-FOLLOW states instead of the entire 2-D state space.
   (b) The action space consists only of two actions to explore at each state.

Disadvantages

1. May take many trials to converge if the environment is too complex.

2. Need for empirical calculation of the parameters in 9.1 that work for any environment. This requires running the algorithm over many different environment configurations and tuning these parameters.

In conclusion, the thesis explores the connection between biological reinforcement learning and its computational counterpart, and uses this link to solve a simple navigation problem.

REFERENCES

[1] A. Mahadevuni and P. Li, "Navigating mobile robots to target in near shortest time using reinforcement learning with spiking neural networks," in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 2243–2250, May 2017.

[2] S. Thrun, "Toward robotic cars," *Commun. ACM*, vol. 53, pp. 99–106, Apr. 2010.

[3] S. Sand, S. Zhang, M. MÃijhlegg, G. Falconi, C. Zhu, T. KrÃijger, and S. Nowak, "Swarm exploration and navigation on mars," in *2013 International Conference on Localization and GNSS (ICL-GNSS)*, pp. 1–6, June 2013.

[4] P. H. T. Kruger, S. Nowak, "Towards autonomous navigation with unmanned ground vehicles using lidar," *Proceedings of the 2015 International Technical Meeting of The Institute of Navigation*, pp. 778–788, Jan. 2015.

[5] B. Tribelhorn and Z. Dodds, "Evaluating the roomba: A low-cost, ubiquitous platform for robotics research and education," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 1393–1399, April 2007.

[6] S. Bahadori, D. Calisi, A. Censi, A. Farinelli, G. Grisetti, L. Iocchi, D. Nardi, and G. D. Tipaldi, "Autonomous systems for search and rescue," 10 2017.

[7] S. Goyal and P. Benjamin, "Object recognition using deep neural networks: A survey," *CoRR*, vol. abs/1412.3684, 2014.

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

[9] G.-S. Yang, E.-K. Chen, and C.-W. An, "Mobile robot navigation using neural q-learning," in *Proceedings of 2004 International Conference on Machine Learning and Cybernetics (IEEE Cat. No.04EX826)*, vol. 1, pp. 48–52 vol.1, Aug 2004.

[10] G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics*. New York, NY, USA: Cambridge University Press, 2nd ed., 2010.

[11] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, 1986.

[13] B. Blaus, "Multipolar Neuron." `https://commons.wikimedia.org/wiki/File%3ABlausen_0657_MultipolarNeuron.png`.

[14] Laurentaylorj, "Action Potential." `https://upload.wikimedia.org/wikipedia/commons/9/95/Action_Potential.gif`.

[15] Chris73, "Action Potential." `https://en.wikipedia.org/wiki/File:Action_potential.svg`.

[16] A. L. Hodgkin and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of Physiology*, vol. 117, no. 4, pp. 500–544, 1952.

[17] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I. homogeneous synaptic input," *Biological Cybernetics*, vol. 95, pp. 1–19, Jul 2006.

[18] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on Neural Networks*, vol. 14, pp. 1569–1572, Nov 2003.

[19] J. Sjostrom and W. Gerstner, "Spike-timing dependent plasticity," *Scholarpedia*, vol. 5, no. 2, p. 1362, 2010. revision #151671.

[20] E. L. Thorndike, "Animal intelligence: An experimental study of the associative processes in animals.," *The Psychological Review: Monograph Supplements*, 2010.

[21] E. M. Izhikevich, "Solving the distal reward problem through linkage of stdp and dopamine signaling," *Cerebral Cortex*, vol. 17, no. 10, pp. 2443–2452, 2007.

[22] M. Egerstedt, "Controls for the masses [focus on education]," *IEEE Control Systems*, vol. 33, pp. 40–44, Aug 2013.

[23] N. Fremaux, H. Sprekeler, and W. Gerstner, "Reinforcement learning using a continuous time actor-critic framework with spiking neurons," *PLoS Comput Biol*, vol. 9, p. e1003024, Apr 2013. PCOMPBIOL-D-12-00983[PII].

[24] M. Beyeler, K. D. Carlson, T.-S. Chou, N. Dutt, and J. L. Krichmar, "Carlsim 3: A user-friendly and highly optimized library for the creation of neurobiologically detailed spiking neural networks," in *2015 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, July 2015.