DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

DEEP LEARNING BASED VISUAL NAVIGATION IN INDOOR ENVIRONMENTS

by

Berk AĞIN

February, 2021

İZMİR

DEEP LEARNING BASED VISUAL NAVIGATION IN INDOOR ENVIRONMENTS

A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Degree of Master of Science in Electrical and Electronics Engineering Program

by

Berk AĞIN

February, 2021
İZMİR

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled **"DEEP LEARNING BASED VISUAL NAVIGATION IN INDOOR ENVIRONMENTS"** completed by **BERK AĞIN** under supervision of **ASSOC. PROF. DR. GÜLESER KALAYCI DEMİR** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Güleser KALAYCI DEMİR

Supervisor

Assoc. Prof. Dr. Ahmet ÖZKURT

(Jury Member)

Assoc. Prof. Dr. Savaş ŞAHİN

(Jury Member)

Prof.Dr. Özgür ÖZÇELİK

Director
Graduate School of Natural and Applied Sciences

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Assoc. Prof. Dr. Güleser Kalaycı Demir for her guidance, support, constructive feedback, endless tolerances, and patience from the beginning to the end of this thesis. I really appreciate for all contributions to thesis and me.

I want to express my gratitude towards the all professors of Electrical and Electronic Department at the Dokuz Eylul University. I am so grateful for professors of Control and Automation Department Assoc. Prof. Dr. Ahmet Özkurt, Asst. Prof. Dr. Hatice Doğan and Dr. Neslihan Avcu for all their supports. I would like also to thank Asst. Prof. Taner Akkan and Asst. Prof. Şükran Kara for all for making important contributions to my professional life.

I am grateful to my colleagues from Turkish Air Force Academy Col. Nuri Gökmen Karakiraz, Asst. Prof. Musa Nurullah Yazar, Asst. Prof. Pınar Öztürk Özdemir, Asst. Prof. Deniz Özenli, Asst. Prof. Gökhan Altın, Maj. Tufan Koç, Res. Asst. Selen Geçgel, Res. Asst. Özgün Devrim Kılıç and all other workmates (Res. Asst. Osman Dağlı, Res. Asst. Ukte Öner, Res. Asst. Veli Uysal, Res. Asst. Ahmet Tuğrul Bakır) for all their assistance, patience and supports during this process.

I want to special thanks to my friends Baki Akgedik, Berkay Muslu, Oğuz Kabakçı and Alper Can İnce for their support, motivation and friendship. I would like to also thank my classmates Mert Şen, Salih Ünsal and İbrahim Demir for all their help, friendship and patience.

Finally, I have deepest gratitude towards to my mother Şengül. I am grateful to my mother, who is the hero behind my achievements, for being with me and supporting me in good and bad times throughout my life.

Berk AĞIN

DEEP LEARNING BASED VISUAL NAVIGATION IN INDOOR ENVIRONMENTS

ABSTRACT

Deep learning methods are used various areas in recent years. Quite efficient results are obtained in recent studies with combination of deep learning and reinforcement learning. Deep learning based reinforcement learning especially gives powerful solutions for complex robotic tasks like navigation. Mobile robots gains new skills with Deep Reinforcement Learning (DRL).

In this thesis, we propose a deep reinforcement learning approach to complex navigation tasks in indoor environments. We chose an unmanned ground vehicle as an agent and performed visual navigation simulation with real-world camera images. Proximal Policy Optimization (PPO) is policy update method which we used. We investigated various kind of neural network models to find best function approximator such as Convolutional Neural Networks (CNN), Multi-layer Perceptron (MLP), Extreme Learning Machines (ELM), Residual Neural Networks (ResNet) and Neural Ordinary Differential Equations (ODEs). Up to the our knowledge, the use of ODEs with DRL in navigation applications has not proposed in the literature. Results show that ODE based DRL algorithm performs well and makes gain the capability of navigation to the agent in indoor environment.

Keywords: Visual navigation, deep reinforcement learning, ordinary differential equations

KAPALI ORTAMLARDA DERİN ÖĞRENİM TABANLI GÖRSEL NAVİGASYON

ÖZ

Derin öğrenme yöntemleri son yıllarda çeşitli alanlarda kullanılmaktadır. Derin öğrenme ve pekiştirmeli öğrenme kombinasyonu ile yapılan son çalışmalarda oldukça verimli sonuçlar elde edilmektedir. Derin öğrenmeye dayalı pekiştirmeli öğrenme, özellikle navigasyon gibi karmaşık robotik görevler için güçlü çözümler sunmaktadır. Mobil robotlar, Derin Pekiştirmeli Öğrenme (DPÖ) ile yeni beceriler kazanmaktadır.

Bu tezde, kapalı ortamlardaki karmaşık navigasyon görevlerine derin bir pekiştirmeli öğrenme yaklaşımı önerilmiştir. Ajan olarak insansız bir kara aracı seçilmiş ve gerçek dünya kamera görüntüleri ile görsel navigasyon simülasyonu gerçekleştirilmiştir. Proksimal Politika Optimizasyonu (PPO), kullandığımız politika güncelleme yöntemidir. Evrişimli Sinir Ağları (ESA), Çok Katmanlı Sinir Ağları (ÇKSA), Aşırı Öğrenme Makineleri (AÖM), Kalıntı Sinir Ağları (KSA) ve Nöral Diferansiyel Denklem Ağları (NDDA) gibi en iyi işlev yaklaşımlayıcısını bulmak için çeşitli sinir ağı modellerini araştırdık. Bilgimize kadar, DPÖ ile NDDA navigasyon uygulamalarında birlikte kullanılması literatürde önerilmemiştir. Sonuçlar, NDDA tabanlı DPÖ algoritmasının iyi performans gösterdiğini ve iç ortamdaki ajana navigasyon yeteneği kazandırdığını göstermektedir.

Anahtar kelimeler: Görsel navigasyon, derin pekiştirmeli öğrenme, adi diferansiyel denklemler

CONTENTS

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

CHAPTER ONE

INTRODUCTION

Machine Learning (ML) is one of the most popular topics of recent times. It is widely used in different areas. Figure 1.1 shows that usage areas of three main branches of ML. Supervised Learning, Unsupervised Learning and Reinforcement Learning (RL) contains plenty of ML algorithms that are applied in different areas. Usage of RL has been also spreading to plenty areas such as robotics, entertainment, industry, education, military, economy etc. (Derrick Mwiti, 2020).

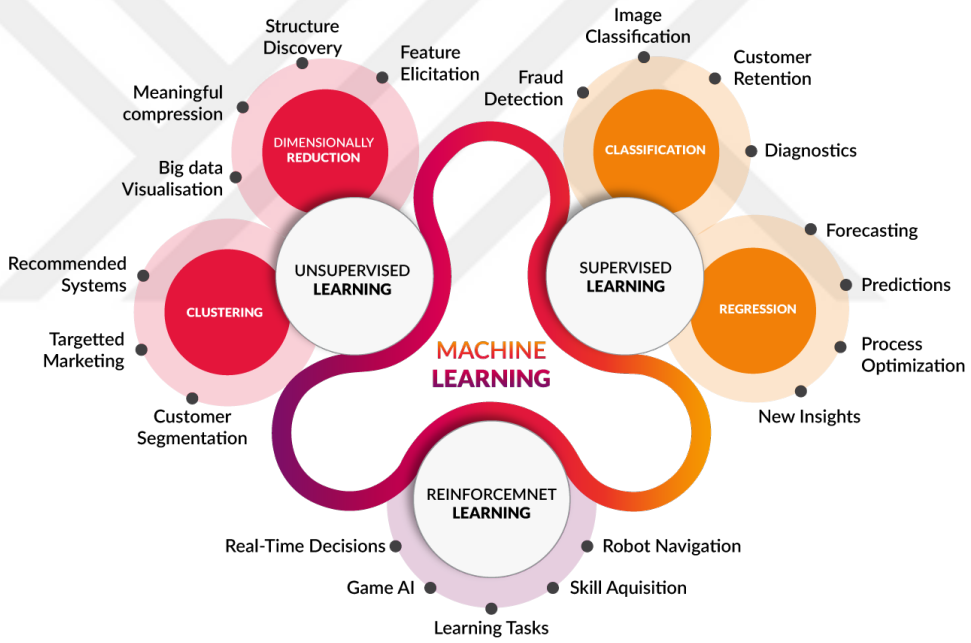


Figure 1.1 Machine learning usage areas (COGNUB Decision Solutions, 2019)

RL approach is inspired from behaviour of nature. It is based on trial and error experiments (Neftci & Averbeck, 2019). In robotics, the robot which is called as agent learns like a child or animal. The agent takes reward and penalty when it performs good actions and bad actions respectively. First aim of the agent is obtaining maximum reward during performing.

RL algorithms take valuable roles especially in video games and robotics with combining deep neural networks. Thanks to computation of deep neural network and simple RL algorithms, Deep Reinforcement Learning (DRL) methods provides more successful and effective results on recent studies. DRL is the good option for solving complex and decision-making tasks in lots of different areas such as video games and robotics. One of the significant study on Atari games with superhuman level was been published by (Mnih et al., 2015). They solved the complex decision-making problems with Deep Q-Networks (DQN). Another important study is AlphaGo which defeats a human world champion in Go (Silver et al., 2017). DRL techniques effect also robotic applications. It makes possible that artificial intelligence agents can act autonomously and accomplished difficult tasks (Arulkumaran et al., 2017). Agents are able to do objectives and auxiliary tasks in 3D partially observable environments successfully (Beeching et al., 2019). DRL approach makes possible to use high-dimensional input images for navigation (Jaderberg et al., 2016; Mirowski et al., 2017). These methods led to an another important study is published as asynchronous advantage actor-critic (A3C) algorithm to improve performance of the agent (Mnih et al., 2016).

Deep Learning (DL) is state-of-art technique gives more accurate results than traditional neural network algorithms (Alom et al., 2019). The word *Deep* is based on the number of hidden layers used in the network. DL contains nonlinear transformation units in order to extract feature of input data. It is used various purpose like segmentation, classification, recognition and prediction. Neural network structures have also gained diversity with the development of deep learning algorithms such as Convolutional Neural Networks (CNN) (LeCun et al., 2015), ResNet (He et al., 2016), Alexnet (Krizhevsky et al., 2012), GoogleNet (Szegedy et al., 2015), Neural Ordinary Differential Equations (ODE) (Chen et al., 2018). Towards the end of our study, a recently published article shows that ODE networks provide an efficient and accurate model for dynamic modeling in continuous control (Alvarez et al., 2020).

We frequently encounter autonomous mobile robots in our daily lives. Mobile robots make a great contribution in areas such as search and rescue, trade, industry,

transportation and exploration. So that, navigation tasks are essential for these robots. To navigate, the robot needs a perception of the environment and a plan based on that perception. Depending on the quality of the plan, the robot creates a path on the map and reaches its destination. This path can be decided with using traditional algorithms such as simultaneous localization and mapping (SLAM) (Thrun, 2002) or path planning approach (Ge & Cui, 2000; Ge et al., 2005). In recent years, there are lots of studies which use DRL approach complex tasks like autonomous driving, self-driving cars, locomotion and navigation. DRL approach has recently presented a different perspective to the navigation problem. It is used for visual navigation and performed in simulation environment with real-world images (Zhu et al., 2017). There are also real world applications which are used RL based techniques Partially Observable Markov Decision Process (POMDP) for visual navigation (López et al., 2003; Ocaña et al., 2005). Recent studies stand out about end-to-end navigation strategies are published for mobile robots (Chiang et al., 2019; Faust et al., 2018; Shi et al., 2020).

There are effective methods for improving performance of navigation such as policy based learning, reward shaping and Curriculum learning (CL). Although, mobile robot has no information about map can occurs a policy which leads to target. Visual navigation uses images coming from the camera placed on robots to create this policy. Methods which are used in (Schulman et al., 2015a) and (Schulman et al., 2017) are good approach to navigation problem. A good reward shaping is significant for RL algorithms. As some researches show that reward shaping could make opportunity to solve complex tasks with little arrangements (Grzes, 2017; Marom & Rosman, 2018; Peng et al., 2018). Solving decision-making tasks may be tough in complex problems. In the educational process, the student should start learning with simple tasks. As the simple tasks are completed, the more difficult tasks train the student and this leads to successful results. CL is inspired by human learning activities and applied to artificial intelligence researches. Combination of CL and RL provides effective and accurate results in training process (Hacohen & Weinshall, 2019).

Simulation takes visible role in robotic researches. It has plenty of benefits like saving time, opportunity of performing countless experiment and cost saving. So, sim-to-real studies spread out in RL applications in recent years (Zhao et al., 2020). In this thesis, we performed the navigation of the mobile robot in a virtual environment developed by Stanford University named *Gibson Environment* (Xia et al., 2018). This virtual environment based on *PyBullet* which is real-time physics simulation (Couman, 2017). Virtual environment includes real building indoor scenes and provides simulation with these images. There are some different studies with various tasks on Gibson Environment in recent years such as navigation with using graph neural networks (Chen et al., 2019), designing neural network for controlling motion policy in navigation (Meng et al., 2019), scaling local control for navigation (Meng et al., 2020), visual model predictive control-policy learning for avoiding from obstacles during navigation (Hirose et al., 2019).

In this thesis, we aimed to navigate a mobile robot in indoor building scenes with using deep reinforcement learning algorithm. We used a virtual environment for simulation. The studies performed in this thesis can be grouped into two components namely, "Visual Navigation" and "Reinforcement Learning". Figure 1.2 shows roadmap and sub-components of these study.

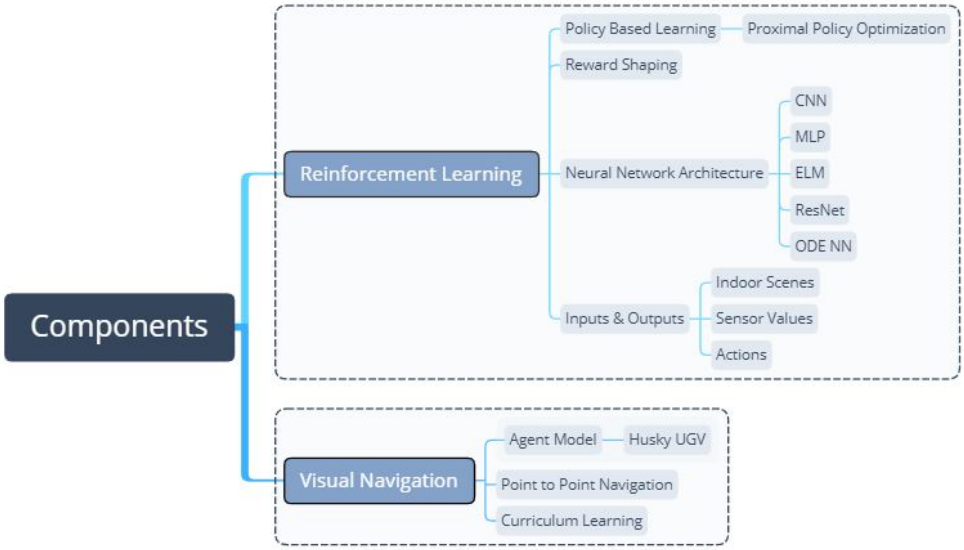


Figure 1.2 Components of our DRL based visual navigation studies

In first section, we applied RL algorithms to the agent model with different kind of deep neural networks. Then, we performed visual navigation with learning principles on virtual environment at second section. We applied various neural network architectures namely CNN, ResNet, Extreme Learning Machines (ELM) and ODE Neural Networks for keeping agent's policy to solve tasks. To control policy, we used Proximal Policy Optimization (PPO) in this study. Since reward shaping is important for a reliable policy, we identified and managed the reward functions carefully for the policy learning of the agent. Then, we defined navigation task in a real building and determined requirements for mobile robot to accomplished goals. A robot model is chosen as *Husky* Unmanned Ground Vehicle (UGV). We prepared waypoints for point to point navigation in the curriculum. We carried out all algorithm developments and improvements for visual navigation through Gibson Environment and used important libraries such as *Tensorflow* and *OpenCV* together with the Python programming language. As importance of this study can be considered as comparison of different neural network models for DRL and being decided to the best model for navigation task on virtual environment. Results show that the best model between applied neural network models can be considered as ODEs neural network model for policy of agent purpose of navigate on indoor scenes.

CHAPTER TWO

REINFORCEMENT LEARNING

2.1 Reinforcement Learning

The first main component of diagram in Figure 1.2 is Reinforcement Learning (RL) architecture. Reinforcement learning is an efficient algorithm for robot navigation applications. Typically, reinforcement learning structure contains two main argument as an agent and environment. The main approach is that the agent should create a policy to reach goal point. Figure 2.1 illustrates main idea of reinforcement learning and working principle with policy based.

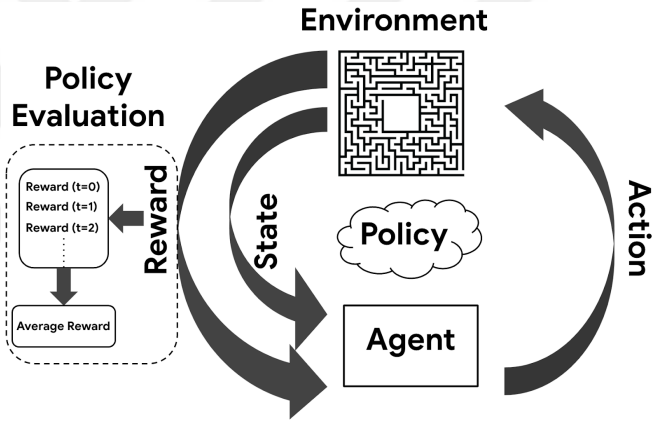


Figure 2.1 General structure of RL (Ali Mousavi, 2020)

In the following we give some of the important terms definitions for the clarity and completeness of the study:

- **Action(A):** All the possible moves that the agent can move in the environment.
- **State(S):** Current situation of the agent returned by the environment.
- **Reward (R):** A value which says that the last action of the agent is good or bad returned by environment.
- **Probability (P):** It is the probability of action in state at time t.

- **Policy (π):** The strategy that the agent believes the best action on the state which is returned by environment.

- **Discount Factor (γ):** The discount factor allows to value short-term reward more than long-term ones. It is used interval [0-1].

- **Value (V):** The expected of long-term returns. $V_\pi(s)$ is based on policy and gives the expected long-term returns of current state.

- **Q-value(Q):** It is very similar to $V_\pi(s)$. In addition to it, there is a action that on the current state based on policy as long-term expected returns. $Q_\pi(s,a)$ contains that state and action pair.

- **Time step:** It means a step of the agent on the environment. Time step is directly related with simulation time of Pybullet in this study.

- **Episode:** The agent takes immediate reward when it interacted with environment. It observes $s_t \in S$ and chooses an action $a \in A$. Then, current state passes next state $s_{t+1} \in S$. The agent reach target point or terminal state, process is ended. This process defined as episode.

- **Iteration:** It represents updating parameters of the learning algorithm number of times.

The model means that simulation which is the dynamics of environment. It contains that probability of actions at current state and next state. In our study, we used model-free reinforcement learning instead of model-based reinforcement learning. Model-free algorithms has no transition matrix and it is based on trial and error experience to update its knowledge. In difference from model-based structure, it does not need to store state and action pairs.

We also decided to use off-policy methods for learning policy. Because, the agent has no any prior information about the environment and it is not possible to create target policy to use on policy method. In off-policy based methods, learning approach focuses that the best policy using trajectories from environment. It is feeded from

another policies. On the other hand, on-policy based methods usually uses a bias on the current policy and updates its policy.

Other feature of the study is having online learning. There are two types of learning approach in learning process as online-learning and offline-learning. The offline-learning has limited information about environment. Online learning is more generalized than offline one since there is no limited information contrast to offline one. In online learning, the agent gains that experiences from environment while it is learning. The agent also comes with exploration/exploitation dilemma in the environment. When it tried new experiences, it can store and reprocess later in next episodes. The exploration-exploitation dilemma is a tough problem for the agent (Thrun, 1998). Exploration is trying to gain more rewards from environment for next states while exploitation is maximizing the expected return given the current states. So that, it must make a trade-off between two options and assign a good strategy for it. We tried to solve this dilemma with using reward shaping and tuning hyper parameters of the algorithm.

2.1.1 Policy Based Learning

In the study, solution of the navigation problem is defined with policy based learning. The agent has no any prior knowledge about map and environment dynamics. Any model does not exist for the states. The agent explores the states, obtains observations from environment during the episode then it improves its policy according to the state values. We have time-step limit for an episode and the agent tries to find best policy π in order to solve navigation task.

The agent which we used in simulation, began with initial state s_0. It picked up an action and moved to next state s_{t+1}. Immediate reward was observed by the agent and next state changed current state at time t. The agent repeated this process until the episode is ended. Observations are called as trajectory the agent collects in one episode with finite time steps. Length of trajectory (ζ) is T as in Equation (2.1).

$$\zeta = S_0, A_0, R_1, S_1, A_1, R_2, ... R_T \tag{2.1}$$

The effect of discount factor γ plays a very important role for collecting maximum rewards in the episode. Because, current states and near states are more valuable than future states in generally. If we want to get the agent to its destination, we should think the best action on states even if the state has low reward in near future. So that, the agent can maximize rewards in long run. This is called as expected return G_t as Equation (2.2).

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

To maximize future rewards in the environment, our agent considered its actions good or bad for next states. This evaluation criteria can be called as value function. Value function of the state under policy which is created by the agent determines the value of that state. Definition of value function under policy π is given in Equation (2.3).

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} R_{t+k+1}|S_t = s] \tag{2.3}$$

Our trajectories contain set of expected returns in the episodes and optimal policy is the best option for our agent. To solve the task, the agent found optimal policy with every iteration. If a new policy is better than previous one, it takes new policy. Optimal policy is defined by optimal state-value function which is given in Equation (2.4).

$$V_*(s) = \max_\pi V_\pi(s) \text{ where } \forall s \in S \tag{2.4}$$

We can define value function with action pairs. Action-state value function keeps action and expectation of return depends to it under the policy π in Equation (2.5).

$$Q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{2.5}$$

If we expand value function with best action for current state, we define optimal action-value as given in Equation (2.6).

$$q_*(s,a) = \max_{\pi} q_\pi(s,a) \text{ where } \forall s \in S \text{ and } \forall a \in A \tag{2.6}$$
$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma V_*(s') | S_t = s, A_t = a]$$

This action-value function leads us to *Belmann optimality equation* which is defined in Equation (2.7):

$$\begin{aligned} V_*(s) &= \max_{a \in A(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(s') | S_t = s, A_t = a] \\ &= \max_a \sum_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma v_*(s')] \\ &= \max_a \sum_{s'} P^a_{s,s'} [R^a_{s,s'} + \gamma \max_{a'} q_{\pi_*}(s',a')] \end{aligned} \tag{2.7}$$

We have also advantage function $A_\pi(s,a)$ to measure better or worse action values on the state. It is defined as the difference between action state value function and state value function. If result of Equation (2.8), $A_\pi(s,a)$ is greater than zero, we can say that the action is good in that state.

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s) \tag{2.8}$$

In order to use returns for long period of time, advantage function can be expanded. Advantage function which contains future reward values is called as Generalized Advantage Estimation (GAE) in Equation (2.9) (Schulman et al., 2015b).

$$\hat{A}_t = \delta_t + (\gamma\delta_{t+1}) + \cdots + (\gamma\lambda^{T-t+1})\delta_{T-1} \text{ where } \delta_t = r_t + \gamma V(s_{t+1}) - V(s_t) \tag{2.9}$$

In our study, the learning process is based on policy. We used Proximal Policy Optimization (PPO) from policy gradient methods (Schulman et al., 2017). Policy gradient methods have some advantages in infinite horizon and continuous space instead of value-based methods. Smooth convergence is observed with policy gradient update. It is also good at stochastic policies. So that, we focused on policy based

methods. Classical policy gradient methods use Equation (2.10) and Equation (2.11) as objective function where $r(\tau)$ is given trajectory.

$$\begin{aligned} J(\theta) &= \mathbb{E}_{\pi}[r(\tau)] \\ \nabla_{\theta} J(\theta_t) &= \mathbb{E}_{\pi}[r(\tau) \nabla \log \pi(\tau)] \end{aligned} \tag{2.10}$$

$$\theta^* = \arg\max_{\theta} J(\theta) \tag{2.11}$$

The policy update to solve the problem is defined gradient ascent or descent. Updating rule of one step with learning rate α is given in Equation (2.12). The figure 2.2 shows that how policy and value converges optimal ones.

$$\theta_{t+1} = \theta_t + \alpha \nabla_{\theta} J(\theta_t) \tag{2.12}$$

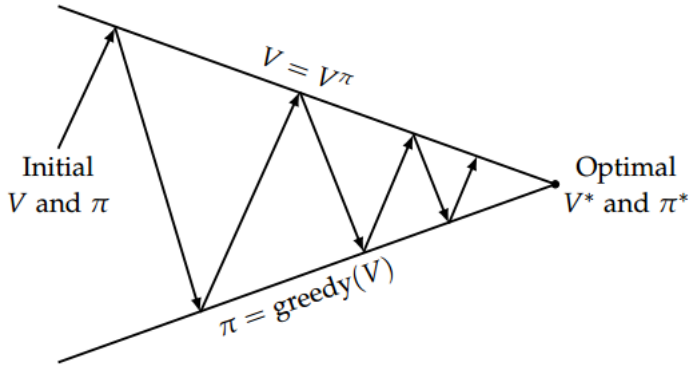


Figure 2.2 Illustration of policy and value iteration (Sutton & Barto, 2018)

PPO is extended version of TRPO (Schulman et al., 2015a). It is effective approach in order to keep under control policy during iterations. It converges quickly and has good performance. 'Importance sampling' and 'KL penalty' are main features of TRPO. In addition to these, PPO uses 'Clipped Surrogate Objective'. Instead of discarding old trajectory data, it is reused for estimating expected values on new policy. Surrogate objective function can be defined as given in Equation (2.13) where $r_t(\theta)$ is the ratio between old and new policies.

$$L(\theta) = \mathbb{E}[r_t(\theta)\hat{A}_{\theta_\pi}(s_t, a_t)]$$
$$r_t(\theta) = \frac{\pi_\theta(s_t, a_t)}{\pi_{\theta_{old}}(s_t, a_t)} \tag{2.13}$$

Kullback-Leibler divergence (KL) is also important for keeping under control the policy updates. In classical calculations, KL divergence represents dissimilarity between two different probability distributions. So, KL divergence measures the difference between two policies in Equation (2.14). Equation (2.14) is substituted into the Equation (2.13) for punishment in case of the big policy update. β parameter is constant for control evaluation. The 'KL Penalty' is defined in Equation (2.15).

$$D_{KL}(\pi_\theta || \pi_{\theta_{old}})[s] = \sum_{a \in A} \pi_\theta(a|s) \log \frac{\pi_\theta(a|s)}{\pi_{\theta_{old}}(a|s)} \tag{2.14}$$

$$L^{KLPEN}(\theta) = \mathbb{E}[\frac{\pi_\theta(s_t, a_t)}{\pi_{\theta_{old}}(s_t, a_t)} A_{\theta_\pi}(s_t, a_t)] - \beta D_{KL}(\pi_\theta || \pi_{\theta_{old}})[s_t] \tag{2.15}$$

In PPO algorithm, we used simpler idea as clipped surrogate function to control policy update instead of KL penalty. Ratio of policies which we mentioned in Equation (2.13) bounded by ε parameter in range of $[1-\varepsilon, 1+\varepsilon]$. So, the final objective function is obtained as Equation (2.16).

$$L^{CLIP}(\theta) = \mathbb{E}[clip(\frac{\pi_\theta(s_t, a_t)}{\pi_{\theta_{old}}(s_t, a_t)}, 1-\varepsilon, 1+\varepsilon)\hat{A}_{\theta_\pi}(s_t, a_t)] \tag{2.16}$$

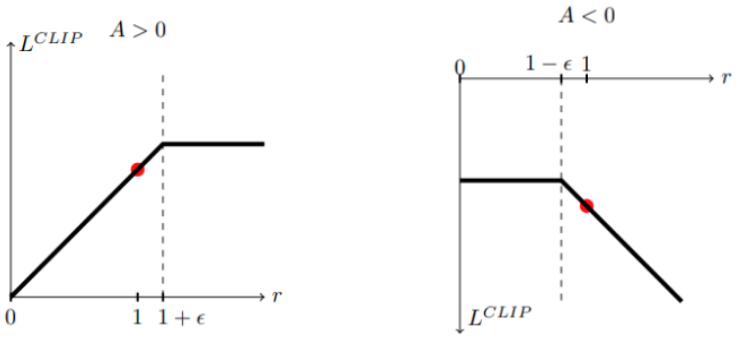


Figure 2.3 Clip surrogate function (Schulman et al., 2017)

Thanks to this boundaries we can control that the policy goes unstable. As it seen in Figure 2.3, According to the sign of advantage value, the objective value is limited by clip parameter. Equation (2.16) was defined as policy surrogate function in our study.

In order to increase exploration diversity, entropy is used in policy based RL. Using entropy loss encourages that increase variance of the agent's actions in probability distribution. We can define the entropy function like in Equation (2.17).

$$H(x) = -\sum_{i=1}^{n} P(x_i) \log_e P(x_i) \tag{2.17}$$

Entropy function can be applied to the long term total rewards to find the optimal policy like as given Equation (2.18). $H(x)$ is defined under policy π and α is used as entropy coefficient to regularize exploration of the agent.

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\pi}[\sum_{t=0}^{\infty} \gamma^t (r_t + \alpha H_t^{\pi})] \tag{2.18}$$

As it seen, objective function uses only policy. It is necessary to minimize error between actual and estimated value of states with using its values. So that, Equation (2.19) can be defined as objective function for values with using mean squared error.

$$L^{VF}(\theta) = (V_\theta - V^{Target})^2 \tag{2.19}$$

We investigated that four parameter to understand behavior of the agent. These parameters were total reward, entropy loss, value function loss and policy surrogate function. We can wrap up all losses in Equation (2.20).

$$L(\theta) = L^{CLIP}(\theta) + L^{VF}(\theta) + \alpha H_t^{\pi} \tag{2.20}$$

2.1.2 Reward Shaping

Reward function is one of the most important part in RL training. We tried to shaping reward function in order to maximize reward calculation which collected in every time step. We used combination of continuous and sparse rewards for complex navigation. Continuous rewards or punishments effected smoothly to cumulative reward. These rewards were defined as continuous rewards such as angle, progress or obstacle distance. On the other hand, we observed that there was a critical balance between reward parameters. If one of them dominated to others, the policy occurred unstable. We defined various parameters as reward and penalty. The parameters used in this study are given in the following:

- (r_t^A): It was called as living reward. It was a sparse reward as 1 or 0. The agent took '1' in case of driving without tipping.

- (r_t^P): This parameter was calculated depend on target distance between two consecutive frames. In case of the agent moved away from target, it took penalty. On the other hand, the agent took reward when it did getting closer to goal point. It encouraged as time cost in episode.

- (r_t^O): It was defined as obstacle penalty. This penalty needed camera images to calculate obstacle distance. Images which taken from render was used for calculation of distance value to obstacle. Distance was converted to penalty value was in interval $[0, -1.350]$.

- (r_t^{ac}): It was defined as angle cost. The purpose of this penalty was that the agent move tend to target point. It was found by calculating between head of agent and target point according to the Euler distance.

- (r_t^{cc}): It was another punishment which was called collision cost for the agent to escape obstacles. when the agent came into contact with obstacles, the number of joints contacted was punished by a penalty coefficient of 0.3.

- (r_t^{sc}): Steering cost was defined as the punishment in case of the agent tries to

turn left or right. This parameter encouraged the agent to proceed as straight as possible. Cost coefficient was 0.1

- (r_t^T): It was target reward when the agent reached to goal point. It was defined as sparse reward as 0.5 or 0. The threshold was used also for distance.

Used reward and penalties of environment dynamics are summed to get total reward function as shown in Equation (2.21).

$$R_t = r_t^A + r_t^P + r_t^O + r_t^{ac} + r_t^{cc} + r_t^{sc} + r_t^T \tag{2.21}$$

Reward shaping had significant effect on navigation progress. It is necessary operation without any model for the environment. In model-free learning, rewards or penalties should be defined appropriately depends on environment dynamics.

2.1.3 Neural Network Architecture

In this study, we have implemented some different kind of neural network architectures to find best neural network model to solve complex navigation tasks. Models were implemented under the reinforcement learning algorithms and the results were observed. It can be said deep neural networks with reinforcement learning perform better policy updating and effective results to navigation problem since computation power.

2.1.3.1 Convolutional Neural Networks

Convolutional Neural Networks are important for neural network architecture. It has crucial role in image recognition, image classification, detection object etc. Input is generally an image. Image contains pixels and the machine processes the image as arrays of pixels. Image occurs from three sizes which are height, width, dimension. If the image is RGB, dimension is 3. Convolution term lies down mathematical expression of convolution which is given in Equation (2.22).

$$[f * g](t) = \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d(\tau) \tag{2.22}$$

Main idea of convolution in neural networks is the same with mathematical expression. A window which is called 'kernel' is swept over on the input image and obtained output image after operation. This output image gives us the feature map of the input one.

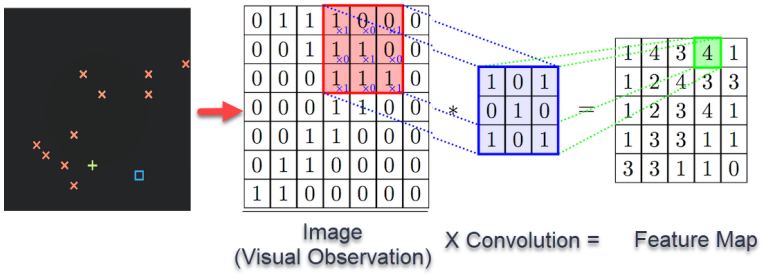


Figure 2.4 CNN operation extracting feature from image (Lanham, 2018)

We explain some important parameters for CNN in the followings for the clarity of the text.

- **Filters:** Dimension of output depends on filter size. Input dimension is $(h*w*d)$ and filter size is (f_h*f_w*d). Feature size is also given in Equation (2.23):

$$(h-f_h+1)*(w-f_w+1)*1 \tag{2.23}$$

- **Strides:** Stride operation can be called shifting operation. Stride defines that how many pixels should be shifted. If stride is unity, the kernel moves only one pixel.

- **Padding:** Padding is significant technique for increasing accuracy in neural networks. Convolution process is completed with all pixels are convoluted by kernel. But, kernel filter performs convolution operation more on middle pixels than edge pixels. This situation causes that the output image contains much information of middle pixels than edge pixels. To reduce this effect, extra pixels are added to image borders. So, the filter moves on edge of image and keep more information which belongs to it. Thanks to padding operation the accuracy of prediction depends on features can be increased.

- **Activation Functions**: Activation functions is used for determining a neuron's output whether true or false. Activation functions can be divided as linear and non-linear functions. Linear activation functions are simple and it can be used on non-complex tasks. On the other hand, non-linear functions are used commonly. It provides on decides of complex problems and output on various data. Sigmoid, hyperbolic tangent, ReLU(Rectified Linear Unit) and Leaky ReLU are most common activation functions. We used the ReLU function in this study.

- **Pooling Layer:** Pooling operation is most common used in neural networks. It reduces size of image and reduce computation of parameters in neural networks. It also can be used to control overfitting. Pooling operation can be called as also downsampling or subsampling. Two function can be defined as most common used pooling operations. These are average pooling and maximum pooling.

- **Fully Connected Layer:** After processes from previous layers, input image is applied to fully connected layer as the flattened matrix. Fully connected layer has same principle with multi-layer perceptron networks. All neurons in previous layers are connected to neurons of the next layer. This connection type may increase accuracy and the computation of parameters. After all layer processes are completed, the network puts the related prediction.

2.1.3.2 Multilayer Perceptron Layers

Multi-layer perceptron (MLP) was used as one of the neural network types which was used in this study. Generally, MLP networks are base of the deep learning applications. It has at least three layer; namely, input, hidden and output layers. In figure 2.5, an example MLP structure is given. According to the complexity of task, numbers of hidden layers can be increased.

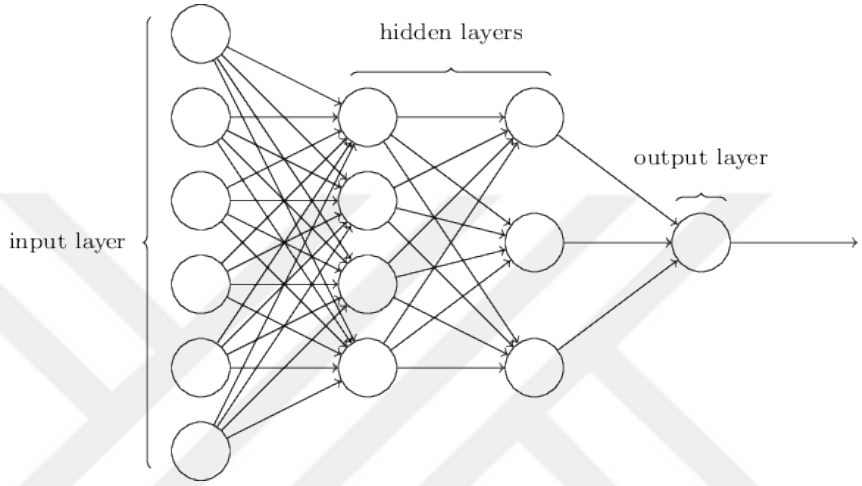


Figure 2.5 Multilayer Perceptron network model (Nielsen, 2015)

Each layer in the networks is represented by Equation (2.24) where y is output, W is weight matrix, x is input and b is bias. In fully connected layers, connections can be set as fully connected. Each neuron is independent and computation of parameter is unique for neuron. As the amount of neurons increases, the training process can slow down.

$$y = Wx + b \tag{2.24}$$

2.1.3.3 Extreme Learning Machines

Combining RL methods and Extreme Learning Machines (ELM) was another important approach for our study (Huang et al., 2006). The visible characteristics of this network are good generalization and fast learning speed. It is a simple idea and has good effects on training process. In Figure 2.6, the neural network architecture of ELM is shown. The steps to implement ELM are given in the following:

- **Set random weights:** The first step is setting random weights matrix and bias value to input layer. These parameters are set also non-upgradable. All input neurons are fully connected to the hidden neurons.

- **Computation of hidden layer:** H is the hidden layer output matrix where K is number units of hidden layer, N number of input samples, $g(.)$ is activation function, x is input vector and b is bias. H output matrix can be calculated in Equation (2.25):

$$\begin{bmatrix} h(x_1) \\ \dots \\ h(x_N) \end{bmatrix} = \begin{bmatrix} g(w_1,b_1,x_1) & \dots & g(w_K,b_K,x_1) \\ \dots & \ddots & \dots \\ g(w_1,b_1,x_N) & \dots & g(w_K,b_K,x_N) \end{bmatrix}_{N\times K} \tag{2.25}$$

- **Activation function:** It is possible to use different activation functions. Non-linear activation functions such as *sigmoid* or $tanh$ can be used in complex problems.

- **Moore-Penrose inverse:** Thanks to this method (Harville, 1997), pseudo inverse matrix of H can be calculated. It also provides generalization of an inverse matrix. In Equation (2.26) pseudo inverse matrix of H is defined:

$$H^+ = (H^T * H)^{-1} H^T \tag{2.26}$$

- β **weight matrix:** β is a special matrix which consists pseudo inverse matrix. Y is target output matrix which has M output neurons. To calculate β the following equation is defined:

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_K^T \end{bmatrix}_{K\times M} \qquad Y = \begin{bmatrix} y_1^T \\ \vdots \\ y_N^T \end{bmatrix}_{N\times M} \tag{2.27}$$

$$\hat{\beta} = H^+ Y$$

- **Output prediction:** In order to predict to output, the algorithm which is given below can be used:

Algorithm 1 Making prediction for output

1: Assigning random weights w_i and bias b_i
2: Calculating hidden layer output H
3: Calculating output weight matrix using $\hat{\beta} = H^{+}Y$
4: Using $\hat{\beta}$ in order to obtain prediction of output where $\hat{Y} = H\hat{\beta}$

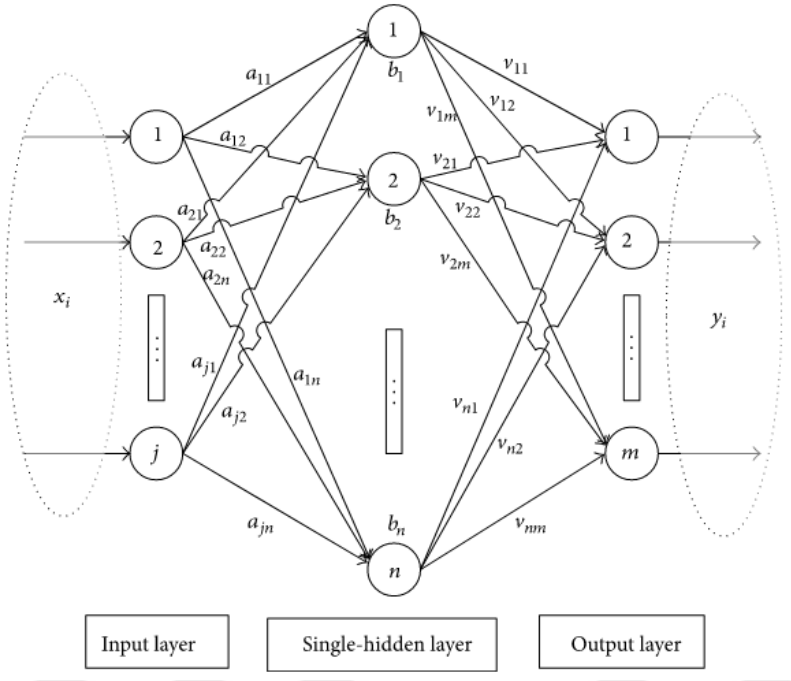


Figure 2.6 Neural network with single hidden layer (Ding et al., 2016)

2.1.3.4 Residual Neural Networks

Residual Neural Networks a novel neural network structure (He et al., 2016). This structure gives more accurate results from traditional convolution neural networks. Their key success is that the proposed neural network architecture gives good results in deeper layers. General problem of CNN architectures is vanishing or exploding gradients. As the neural network deepens, this problem arises. In order to get rid of this problem, 'Skip Connection' is developed in residual block. The identity connection which is given in figure 2.7 provides moving the input to last layer. So that, the input is combined with output and transferred to the next layer.

Equation (2.28) defines that the residual block where x is input and y is output. F function also presents residual mapping for learning.

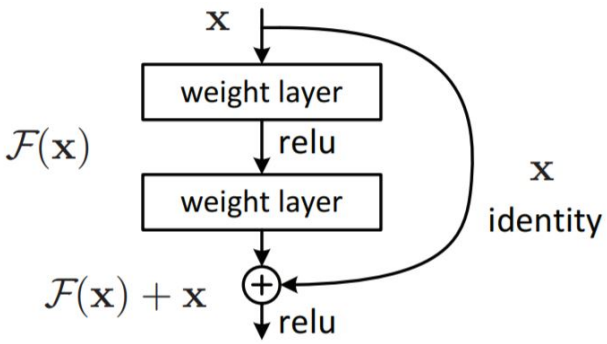


Figure 2.7 The residual block (He et al., 2016)

$$y = F(x, \{W_i\}) + x. \tag{2.28}$$

In case of x and F dimensions are not equal, the projection is required. Equation (2.29) can be used when input and output dimensions are not same where W_s is a square matrix.

$$y = F(x, \{W_i\}) + W_s x. \tag{2.29}$$

ResNet and its method which is used such as skip connection or identity block has some advantages.

- In deeper neural networks, it can be reduced the effect of vanishing gradient problem by adding input to output.
- It can increase the speed of learning in deep neural networks.
- Observing high accuracy on training. Image classification is the significant implementation for ResNet.

2.1.3.5 Neural Ordinary Differential Equations

Neural Ordinary Differential Equations are new approach which try to solve the time series data problem (Chen et al., 2018). Their contributions provide new structure to the neural networks. Neural ODEs build a bridge between physic simulation and real world. It can define continuous system successfully such as real world tasks. In nature, it can be difficult to model data and information. A process is used to model

the data. This process explains the function which tries to find prediction of the data. Machine learning approach helps to describe this function. A standard function can be defined like in Equation (2.30):

$$f : x \longrightarrow y \tag{2.30}$$

Using ordinary differential equation with combining machine learning helps to define interpret data and time series accurately. It is also good solution for dynamic physical problems. Differential equations are suitable to describe time series. Derivative of function shows that the change to approach the desired function. ODEs can be solved by well-known and simple Euler's method. Equation (2.31) defines the Euler's method. The solution is approximated function in discrete time as given in Equation (2.32).

$$\frac{dy}{dt} = f(t,y) \qquad y(t_0) = y_0 \tag{2.31}$$

$$y_{n+1} = y_n + f(t_n, y_n).(t_{n+1} - t_n) \tag{2.32}$$

Neural ODEs can be used instead of Residual Networks. ODEs solution in time series is familiar to ResNet identity block. The chain of residual blocks is a solution of ODEs. ResNet has also initial state like ODEs. $h(0)$ is a hidden state at $t = 0$. Differential function is defined as transformation in layers of networks. Just like time series, series of layer transformation can be solved with ODEs. Residual neural network equation is given in Equation (2.33). It appears like the modelling pattern of an ODEs.

$$h_t = h_{t-1} + f(\theta_t, h_{t-1}) \tag{2.33}$$

Figure 2.8 shows that comparison of Residual Network and ODE Network. As it is seen, transformation of hidden state is smooth on ODE Network. Although, ODE can define continuous time series, neural networks are defined only at natural t. So that, black points which are in the figure are evaluation points at specific time.

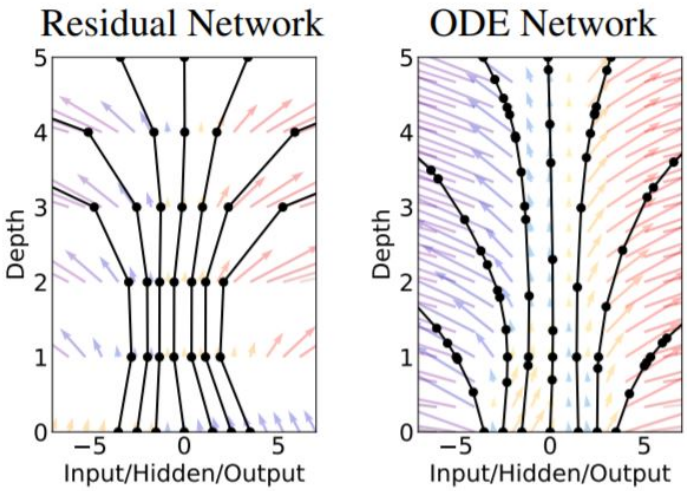


Figure 2.8 Comparison of ResNet and ODE NN (Chen et al., 2018)

The adjoint sensitivity method which is used in study (Chen et al., 2018), provides to reach loss any points in time. It gets with low memory cost and controllable numerical error. Loss function can be defined as given in Equation (2.34):

$$L(z(t_1)) = L\left(z(t_0) + \int_{t_0}^{t_1} f(z(t), t, \theta)dt\right) = L(ODESolve(z(t_0), f, t_0, t_1, \theta)) \quad (2.34)$$

To optimize L function, it is necessary to calculate gradients with respect to θ. Firstly, it needs to be find gradient of loss at $z(t)$ instant. It can be defined as adjoint $a(t) = \frac{\partial L}{\partial z(t)}$. Thanks to this adjoint, the chain rule can be applied to the equation with investigating earlier time values and solved loss which depends on θ like in Equation (2.35). The Figure 2.9 shows that calculating states backwards in time.

$$\frac{dL}{d\theta} = \int_{t_1}^{t_0} a(t)^T \frac{\partial(z(t), t, \theta}{\partial\theta} dt \quad (2.35)$$

There are some advantages of using ODE in neural networks as mentioned in (Ricky T. Q. Chen, Neural Differential Equations, 2018):

1. **Memory Efficiency:** The forward pass computations contain only constant memory cost in depth instead of keeping all parameters while computing gradients with backpropagating.

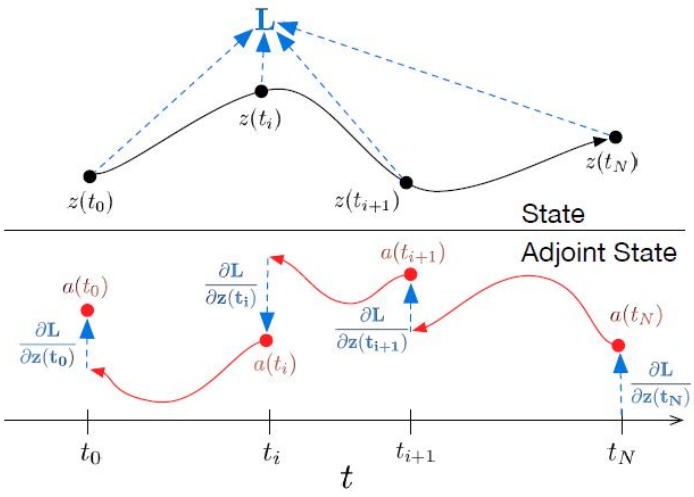


Figure 2.9 Reverse-mode differentiation of an ODE solution (Chen et al., 2018)

2. **Adaptive computation:** Although, it uses very simple and old method which is Euler's method, it can compute error and loss. So, function approximation is found successfully.

3. **Scalable and invertible normalizing flows:** It is side-benefit of Neural ODEs. It is easy to compute and avoids from bottleneck in normalizing flows.

4. **Continuous time-series models:** The quite important advantage is usable for continuous time series. ODEs can reduce effect of interpret data and it is well defined for realistic and dynamic models.

2.1.4 Input & Output Characteristics of the Neural Networks

Quality of navigation depends on quality of input dataset. In our study, we experimented that plenty variation of input dataset. We observed that it was not effective solution using only camera images from environment for neural networks. It was seen clearly proprioseptive sensor data on the agent was efficient on training. Combination of sensor data and camera images gave the best predictions of actions for the task.

As we mentioned in previous sections, we used RGB and depth camera images to navigate without any collision. Also, proprioseptive sensor data supported to these task efficiently. Example of some RGB and depth camera images are given in Figures 2.10 and 2.11. We chose resolution 128 due to limits of hardware on which we performed

training. So, RGB images had $128x128x3$ and depth images $128x128x1$ dimensions. These images were used as state observations which were taken from environment.

(a) RGB frame:165 (b) RGB frame:312 (c) RGB frame:335 (d) RGB frame:350

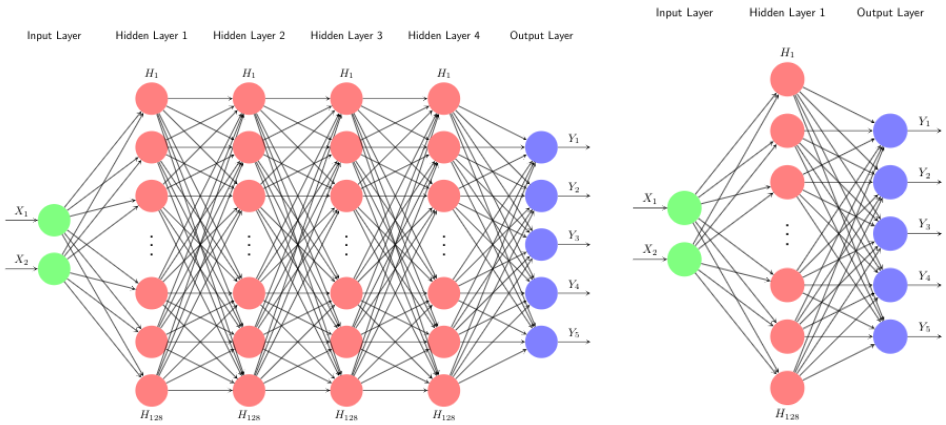
Figure 2.10 RGB frame samples

(a) Depth frame:165 (b) Depth frame:312 (c) Depth frame:335 (d) Depth frame:350

Figure 2.11 Depth frame samples

We also used proprioseptive sensor data as input of neural network. Proprioceptive sensor data on the agent were respectively height difference, vertical and horizontal angle of the agent to the target, linear velocities for each axis in the x-y-z coordinate system, roll and pitch angles of the body, angular velocities on the x-y-z joints, contact values of robot wheels consisting of 8 different joint values that affect the motion on the robot with the ground. The proprioseptive sensor data is represented by $23x1$ dimension vector.

We trained sensor data on neural networks as MLP and ELM. The architecture of MLP and ELM are given in Figure 2.12a and 2.12b respectively. For both of MLP and ELM architectures, There were two inputs (camera and sensor). In case of non-visual training, first input was be discarded. MLP had 4 hidden layers and 128 neurons in each hidden layer in our study. Neurons at neural networks were fully connected to each layer. ELM network is a special form of MLP neural networks. We also built this architecture in our study. It had only one hidden layer and 128 neurons in that layer.



(a) Multi-layer Perceptron Network

(b) Extreme Learning Machines Network

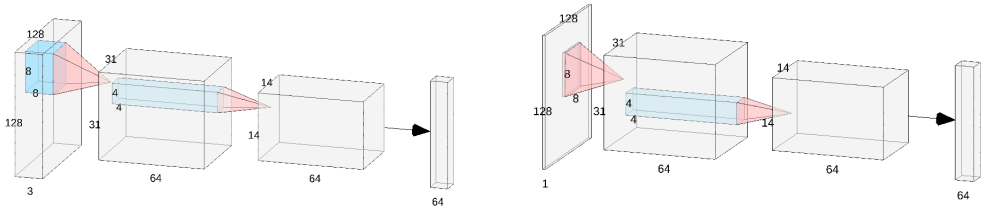
Figure 2.12 Neural networks for sensor data

Outputs of MLP and ELM were defined as actions. The action space was defined as discrete. Each output neuron presented one of the actions in discrete action space. Action space was defined in five $5D$ vector [Forward, Backward, Turn Left, Turn Right, Stop]. Neural networks predicted action values according to the state values and observations. Quality of actions was getting better while the policy was improved.

Observations which are in image form as RGB or depth need convolution operation for DRL. It is important method for extracting feature and merging state values. We implemented it also in our study as a sequential convolution operations with auxiliary deep learning elements. The Figure 2.13 illustrates operation layers. Same parameters were used for both of RGB and depth images. In the first convolution layer, kernel filter size was $8x8$. After input images was operated by first kernel filter, a $4x4$ sized kernel filter operated output image of first layer in second convolution layer. In last convolution layer, $3x3$ kernel filter was used. *ReLu* activation function was implemented after each convolution operation. As a final step, output image of last convolution layer was flattened and created as a vector in dense layer.

It was experimented that using only camera images didn't gave good results. So that, we combined camera images and sensor data for neural networks input. We defined this operation as fusion of data. This approach was implemented as input to

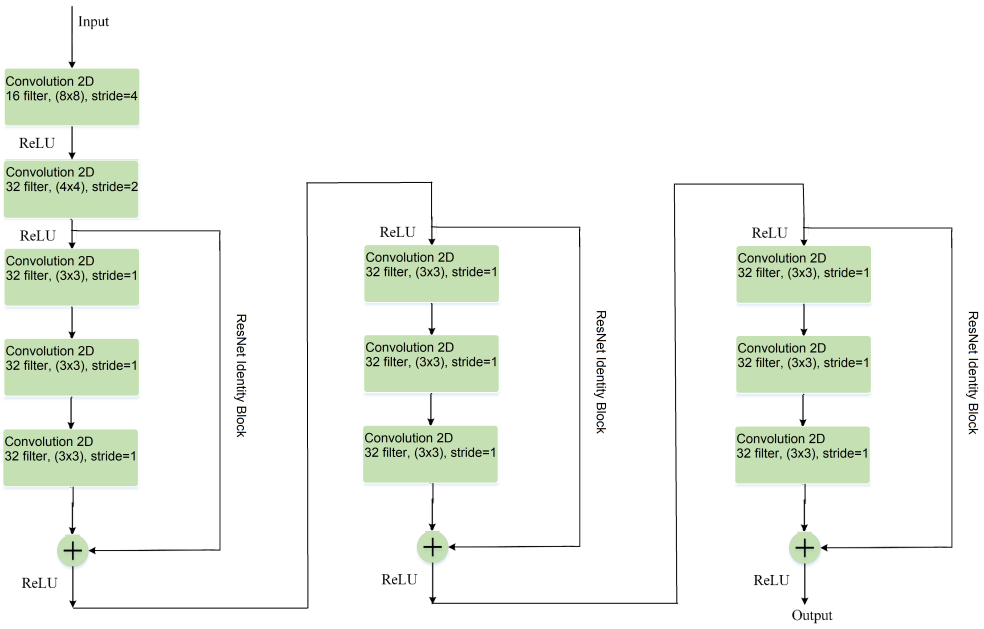
MLP and ELM neural network models. Figure 2.13a and 2.13b represent convolution layers of MLP and ELM respectively.



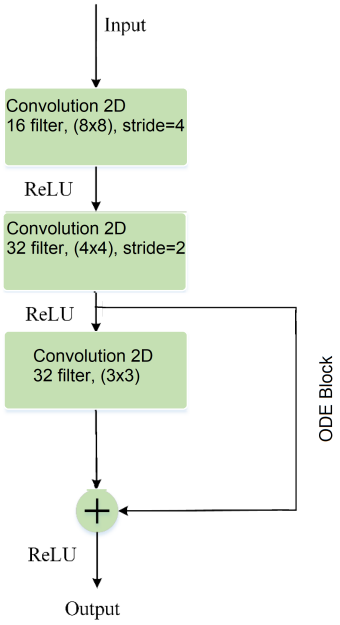
(a) Convolution layers for RGB image (b) Convolution layers for depth image

Figure 2.13 Convolution layers for camera images

Another neural network types which we used were ResNet and ODE neural networks. Following Figures 2.14a and 2.14b illustrates building of neural networks.



(a) Building of ResNet



(b) Building of ODE NN

Figure 2.14 Convolution layers and blocks on ResNet and ODE NN

CHAPTER THREE

VISUAL NAVIGATION

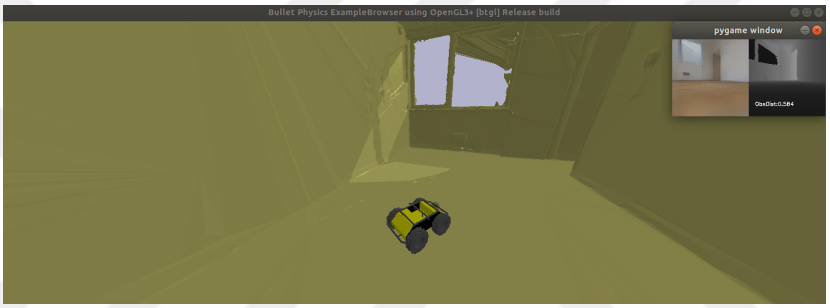
Visual navigation is important component for autonomous agents which perform navigation task. In this chapter, we will explain model of the chosen agent, performed navigation type and importance of the curriculum learning approach for the agent. There are various objectives of the agent in the environment to perform. We can explain the kind of tasks from (Anderson et al., 2018) as followings:

- **Goal Point:** The agent tries to reach the target point which is given in coordinate system. It determines own beginning location and calculates distance between target point and its location. Thanks to sensory inputs, the agent avoids from obstacles and moves towards to target point.
- **Object Goal:** Objective of the agent can be defined such as 'find key', 'find television' or 'take a cup' etc. To perform these tasks, the agent must have knowledge about object. Supervised learning can be used for categorizing and deciding to the object after process of navigation with reinforcement learning algorithms.
- **Area Goal:** In this kind of the task, target points can be defined with specific local area names. For instance, the agent tries to navigate namely 'kitchen' or 'living room' from 'bathroom'. But in order to reach target point, the agent needs knowledge about these areas.

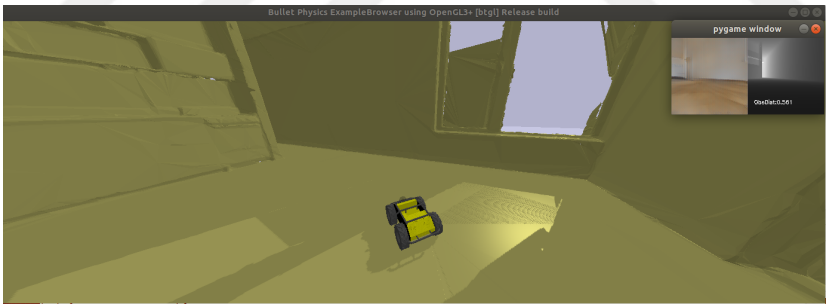
In this study, we aimed point-to-point navigation type via husky robot in virtual environment. The agent planned to navigate starting point to goal point. These points must be different in training phase to generalize the navigation. The agent has a camera and different sensors for sensing environment observations, so that the agent can able to detect objects and avoid from obstacles in navigation process.

We studied on Gibson Environment as virtual environment which contains visual space perceptions (Gibson, 2013). Gibson is based on virtualizing real spaces, rather than using artificially designed ones, and currently includes over 1400 floor spaces from 572 full buildings (Xia et al., 2018). Each virtual environment space is configured with *yaml* files inside of the framework file system.

Local planning and obstacle avoidance are available objectives in order to solve navigation problem. These objectives was performed by the agent in the virtual environment. The agent can able to catch RGB and depth images from environment thanks to its virtual camera. Some screenshots from environment are given in Figure 3.1.



(a) Screenshot 1



(b) Screenshot 2

Figure 3.1 Screenshots of Gibson Environment

Configuration file was created for selected environment in *yaml* format. We defined environment specifications, robot specifications, training parameters, learning methods, user interface parameters and mode parameters in this configuration file.

3.1 Agent Model

This section introduces model of robot which we used in virtual environment for navigation. Gibson Environment contains different agents with Unified Robotic Description Formats (URDF) which based on *PyBullet*. Specifications of model are important for modelling like physical joints, degree of freedom and controller. Some agents which can be used in Gibson Environment are shown in Table 3.1.

Table 3.1 Agents list of the environment

Agent Name	Degree of Freedom	Controller
Mujoco Ant	8	Torque
Mujoco Humanoid	17	Torque
Husky Robot	**4**	**Torque, Velocity, Position**
Minitaur Robot	8	Sine Controller
TurtleBot	2	Torque, Velocity, Position
Quadrator	6	Position
JackRabbot	2	Torque, Velocity, Position

We used Husky UGV as an agent in this study. The agent model can move on virtual space and has capability of the navigation with reinforcement learning in robotic applications. It has large payload capacity and enchancable platform. Some sensor packages can be integrated for different applications such as lidar, manipulator, 3D camera etc. Gibson Environment consists *Pybullet* based physics simulation motor. The husky is modelled with sensors, RGB and depth cameras to perform navigation tasks in this study. A picture of the agent with camera is given in Figure 3.2.

To simulate the model, the kinematic model of the husky was created using the Cartesian coordinate system. Joints of the robot like wheels, body, contact points were also modelled and tested in Physic simulation.

The husky has four wheels like common ground vehicles. It is driven by skid steer motion. To reach the target, the agent needed to calculate angle difference and

Figure 3.2 The agent with RGB camera

Euclidean distance. The Euclidean distance between posture of vehicle (p) and target point (q) is calculated in 3D space as given in Equation (3.1).

$$d(p,q) = \sqrt{(q-p)^2} \tag{3.1}$$

The tending angle for target is defined as the angle between target point and agent head. This angle is used to teach how the agent should be directed to the target point. The schematic diagram of the agent is given in Figure 3.3.

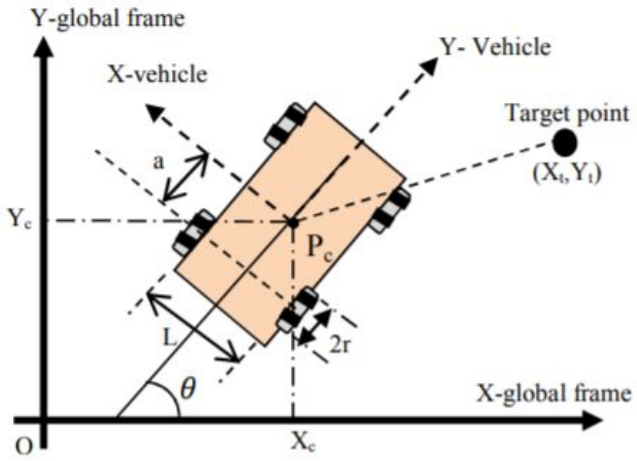


Figure 3.3 The agent Cartesian coordinate according to the target point (Al-Mayyahi et al., 2017)

The husky robot is modelled with five action in discrete action space as forward, backward, turning right, turning left and stop respectively. Actions can be applied with using torque and constant velocity. We didn't prefer continuous action space due to

tasks are getting more complex and have large parameters to compute, although continuous action space is closer to the real-world than discrete action space. Simulation of model is easy to implement in discrete action space and calculations of parameters are less than continuous action space. The implemented discrete action matrix is given in Equation (3.2). Two different variables were implemented to wheels of the agent for movements as torque and velocity in the study. Experiments have shown that the torque variable applied to the wheels has the ability to accelerate faster than the constant velocity variable on the wheels. To move the agent, we applied necessary torques (T) to wheels for given five actions.

$$\begin{bmatrix} Forward \\ Backward \\ Turn_{right} \\ Turn_{left} \\ Stop \end{bmatrix}_{5\times4} = \begin{bmatrix} [T & T & T & T] \\ [-T & -T & -T & -T] \\ [T & -T & T & -T] \\ [-T & T & -T & T] \\ [0 & 0 & 0 & 0] \end{bmatrix}_{5\times4} \tag{3.2}$$

3.2 Point-to-Point Navigation

The main goal of the agent was to reach the target point from the starting point without hitting obstacles in our study. We used waypoints which is available on the building scenes for navigation. The dataset of waypoints consist from data type (test or train), model-id of building, *(x,y,z)* coordinates for start point, yaw angle on scene, *(x,y,z)* coordinates for goal point, Euclidean distance, Geodesic distance and complexity of navigation respectively. These parameters were used by the agent with purpose of computation of the navigation specifications.

Euclidean distance gives that the straight line distance between two different points. On the other hand, Geodesic distance gives shortest available path distance which mobile robot can move. It s possible to calculate navigation complexity thanks to these calculations. Navigation complexity (A) is defined the ratio of Shortest Path (SP) to Straight line Distance (SD) as given in Equation (3.3). We applied randomness to each starting points and goal points in every episodes to encourage the agent faces with different scenes in the environment by increasing exploration.

$$A = \frac{SP}{SD} \tag{3.3}$$

In this study, we chose episodic learning for training, so we used each row in the waypoints dataset as an one episode. In realistic applications, it is impossible to collect all samples at the same time. The agent had time steps limit for exploring and collecting observations from environment. This feature leads to episodic learning. Specifications of the episodic learning which was used in this study can be explained as followings:

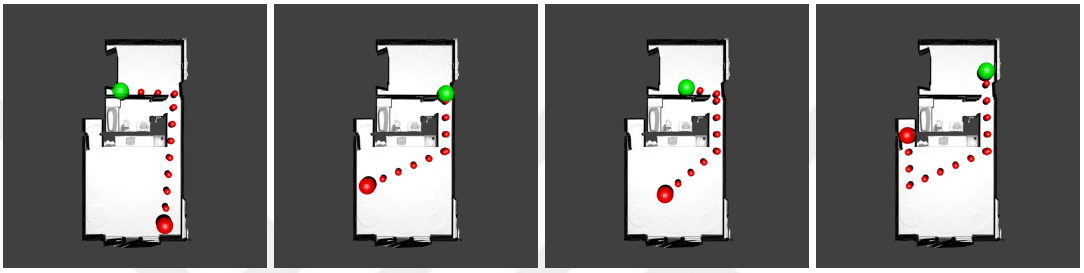
- There was no any model for environment. Model-free learning was applied.
- The agent collected observations during each episode. After the episode ended, policy was updated.
- PPO is an on-policy learning method and effective for episodic problems. We recorded samples from episode in trajectory and policy was updated according to the trajectory.

According to the definition of a navigation task, evaluation method which is called Success Path Length (SPL) was created to measure navigation success of the agent. When the agent crosses the threshold distance to the target point, a signal was generated. This signal contains information about success of episode. Equation (3.4) defines success path length where N is number of episode, l_i shortest path length comes from Euclidean distance, p_i is the total distance which the agent moves and S_i is the binary constant that changes according to the success of the episode. These dataset

$$SPL = \frac{1}{N}\sum_{i=1}^{N} S_i \frac{l_i}{max(p_i, l_i)} \tag{3.4}$$

Initial points and target points take crucial role in point-to-point navigation. As we mentioned in the beginning of the section, complexity of navigation varies according to the these points in the environment. We observed that it was hard to navigate from a room to another one at first iterations of the training. To handle this problem, the agent

learned to read goal points which had low complexity. The agent explored different initial points and goal points in each episodes. The agent improved its policy with different navigation plans consisting of waypoints. The main navigation plan was created from these points partially. In finally, behavior of the agent was made better at states thanks to waypoints after every iteration and the agent gained experience to reach goal points. Some visualization examples of the navigation plan in environment are given in Figure 3.4. Red points in the figure occurs from goal points or starting points in the waypoints dataset.



(a) Navigation Plan 1 (b) Navigation Plan 2 (c) Navigation Plan 3 (d) Navigation Plan 4

Figure 3.4 Example of navigation plans

3.3 Curriculum Learning

Curriculum learning is quite important tool for a neural network learning process (Elman, 1993). This learning strategy is a very simple algorithm that starts with easy tasks and continues with difficult tasks. A 3-years child can not able to read a novel or solve an integration problem. A learning process of child should be cumulative and trace to education life systematically. Training of neural networks can be considered as growing a child. For a successful training, easy tasks should form the basis to next complex tasks. For instance, learning algebra is the first step before calculus.

In reinforcement learning, the agent can be considered as a child. In this thesis, the task of the agent is navigation. Navigation complexity is directly proportional to the ease or difficulty of the task. We defined that navigation complexity as given in Equation (3.3). Agent starts learning point-to-point navigation with waypoints which have navigation complexity that is equal to one. In the next episodes, these complexity increases step by step. Example of waypoints is shown in Table 3.2.

Table 3.2 Waypoints order for CL

split	sceneId	startX	startY	startZ	startAngle	goalX	goalY	goalZ	dist	pathDist	complexity
train	Euharlee	-1.968	3.558	0	3.097	-2.483	-3.391	0	6.968	6.968	1
train	Euharlee	-1.968	3.558	0	3.097	-2.573	-2.53	0	6.118	6.118	1
train	Euharlee	2.087	3.13	0	1.184	-1.918	1.2	0	4.445	4.445	1
train	Euharlee	-1.918	1.2	0	5.229	-2.473	-5.562	0	6.785	6.797	1.002
train	Euharlee	-0.052	3.565	0	2.496	-2.573	-2.53	0	6.596	6.784	1.029
train	Euharlee	2.087	3.13	0	1.184	-2.573	-2.53	0	7.331	7.843	1.07
train	Euharlee	2.087	3.13	0	1.184	-2.57	-4.226	0	8.706	9.539	1.096
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

As it seen from the table, the agent tries to learn easy point-to-point navigation in early steps. There are some points which have same complexity. Though there are some waypoints that have same complexities their scenes are different. So that, the agent can able to face with various scenes with same or different complexity.

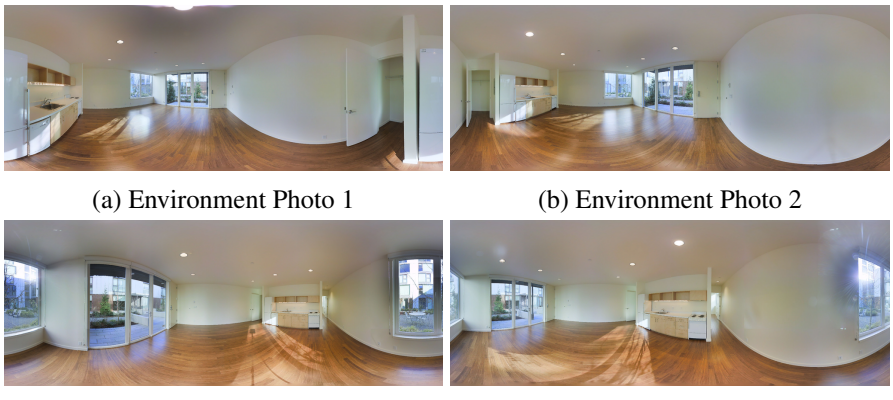
CHAPTER FOUR

APPLICATION & RESULTS

We performed the visual navigation with RL in *Gibson Environment*. In this chapter, the process about training and parameters were explained. During training and test processes, neural network models which contains RGB images used GTX780 as GPU, 64GB RAM and 12 thread 3.60GHz Intel i7-4960X as CPU. On the other hand, models which used depth images or sensor values, used GTX1050 as GPU, 16GB RAM and 8 thread Intel i7-7700HQ as CPU.

4.1 Building Environment

Gibson Environment dataset includes 512 spaces. We took care selecting building which has good rendered scenes. Because, some of the dataset of building had huge holes or blank rendering. We selected two different buildings named 'Euharlee' and 'Aloha'. Good rendering, low scene loading time and reachable areas are the main specifications for learning process. These advantages made up the purpose in selecting these buildings. We used Husky unmanned ground vehicle as an agent and tested the model in simulation environments. The environments have real-world image dataset. Some example panoramic photos from the 'Euharlee' building are given in Figure 4.1:



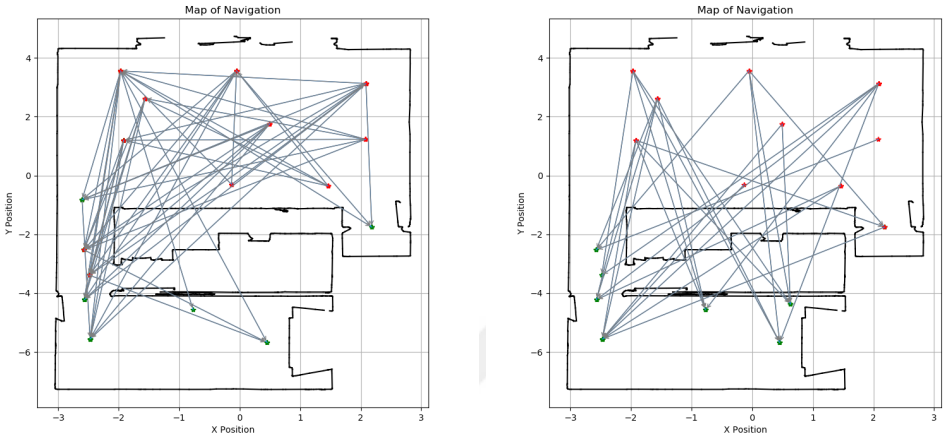
(a) Environment Photo 1 (b) Environment Photo 2

(c) Environment Photo 3 (d) Environment Photo 4

Figure 4.1 Example the panoramic photos of the 'Euharlee' building (Standford Vision Lab, 2018)

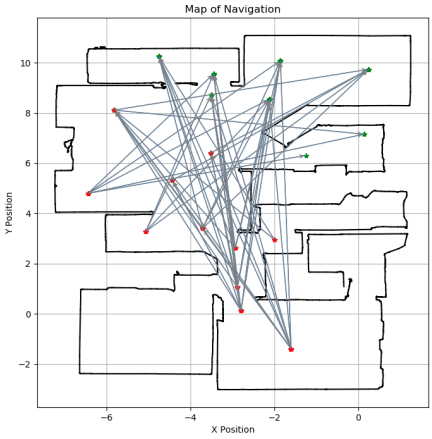
4.1.1 Navigation Scenarios

Our navigation scenarios which we used in training and test experiments in 'Euharlee' and 'Aloha' maps were given in Figure 4.2.

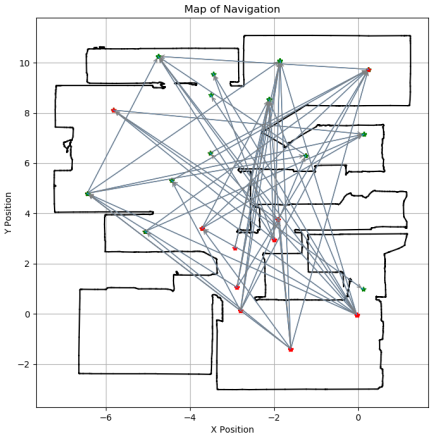


(a) Train set of waypoints in 'Euharlee'

(b) Test set of waypoints in 'Euharlee'



(c) Train set of waypoints in 'Aloha'



(d) Test set of waypoints in 'Aloha'

Figure 4.2 Waypoints for point-to-point navigation

In experiments, 50 different navigation plans for training and 30 different navigation plans for test were used. Although some waypoints has same initial points, they had various navigation complexity due to different goal points.

4.1.2 Reward Function

Reward function is important for the agent in understanding the environment dynamics. We consider several rewards and penalties for the reward function. Then, a balance was established between reward function arguments with testing. Our reward function was given in Equation (2.21). Proprioceptive sensor data dominated to total reward due to arguments in reward function. We tested each reward parameters with 300 timesteps, 100 episodes and 100 iteration. We used MLP network for policy update in following experiment. Table 4.1 shows the effect of combination the arguments of different reward and penalty.

Table 4.1 Results of reward function arguments

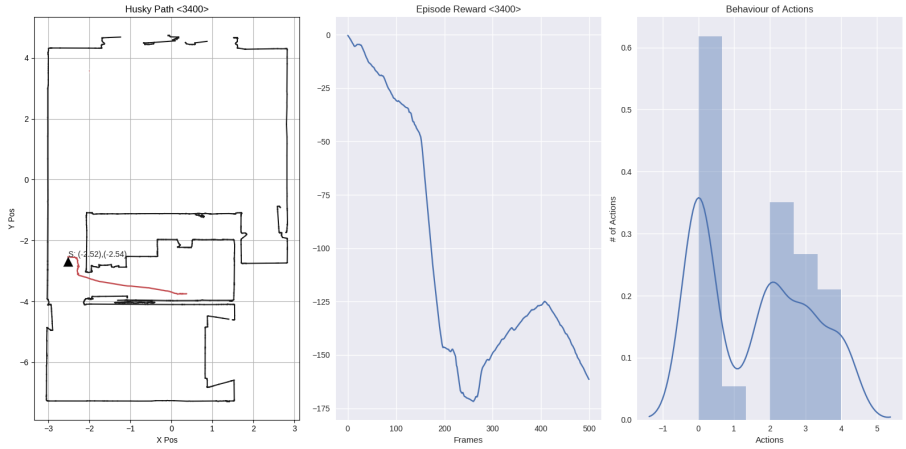
Arguments in reward function	**Total reward at the end of 100^{th} iteration**
r_t^{ac}	-5
$r_t^{ac} + r_t^P$	70
$r_t^{ac} + r_t^P + r_t^T$	80
$r_t^{ac} + r_t^P + r_t^T + r_t^{cc}$	65
$r_t^{ac} + r_t^P + r_t^T + r_t^{cc} + r_t^{sc}$	50
$r_t^{ac} + r_t^P + r_t^T + r_t^{cc} + r_t^{sc} + r_t^A$	100

Positive rewards encouraged to the agent for action which took reward. Also, we observed that the agent stucked in episode without any movements due to high positive reward. The agent behavior was balanced with appropriate rewards and penalties to solve navigation problem in the experiment. Some samples from episodes were given in Figure 4.3.

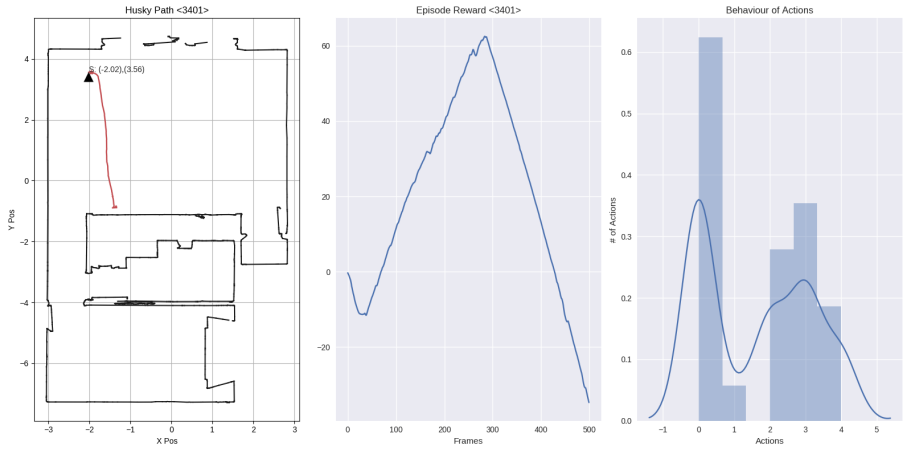
4.1.3 Experiments with Neural Network Models

4.1.3.1 Comparison of Neural Network Models for Sensor Data

We analyzed the effect of ELM and MLP networks in this study. The experiment contained only sensor data and fusion of sensor with depth images for both models. We performed this experiment in 'Aloha' building. Hidden layer size of 128 neurons were used for both hidden layers of ELM and MLP. Training process was observed during 6 iteration and results tabulated in Table 4.2. As it seen from the table, ELM



(a) For Episode=3400



(b) For Episode=3401

Figure 4.3 Reward analysis of the agent

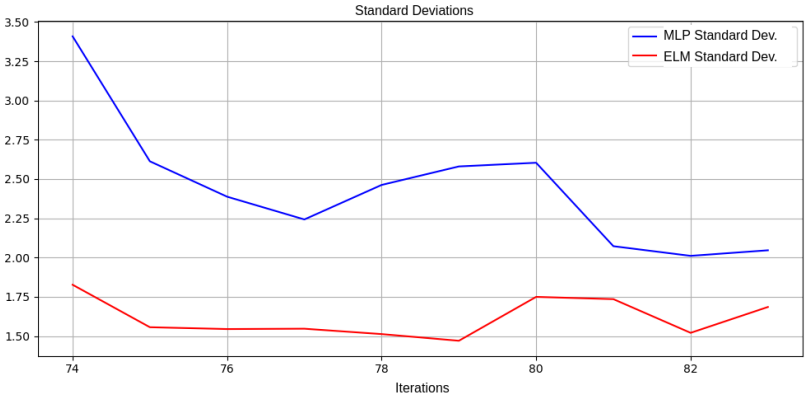
network provided the time advantage in training process almost as much %3.3 as than MLP network. The time advantage was increased to %9.6 when sensor data only was used. When the fusion input which means combination of sensor and depth image data was used, time saving decreased compared to only the sensor input. The reason of this can be explained as the convolutional neural network structure used for depth images shows a more dominant characteristic on the layer that receives the sensor information.

Table 4.2 Training time for ELM and MLP networks with different inputs

Neural Network Model	**Inputs**	**Time**
ELM	Sensor+Depth	10 m 32.51 s
ELM	Sensor	6 m 43.57 s
MLP	Sensor+Depth	10 m 54.06 s
MLP	Sensor	7 m 26.41 s

Total cumulative reward with 85 iterations and standard deviations of it at last 10 iterations were given in Figure 4.4a and 4.4b. In this experiment, sensor data only was used to observe the effect of ELM network clearly.

(a) Total Reward of ELM and MLP



(b) Standard Deviations for ELM and MLP in last 10 iteration

Figure 4.4 Results for ELM and MLP networks

It can be said that ELM network has quicker training time compared to MLP network with success results in navigation problem. The agent obtained converged reward value at approximately 58. iteration for MLP network. On the other hand, converged reward value was seen at 16. iteration for ELM network. In Figure 4.5, successful point-to-point navigation was shown in different episodes with ELM neural network model. These results were presented as paper in (Agin & Demir, 2020).

Another experiment to compare ELM and MLP models was performed in 'Eu-harlee' building map. The experiment had 500 timesteps, 50 episode and 150 iterations with $3.75M$ total frame. According to the observed results MLP network was more successful than ELM network with given environment dynamics. On the other hand, ELM network completed to training process faster than MLP network again. The training times were recorded as T_{ELM} = 4 hour 35 minutes and T_{MLP} = 5 hour 18 minutes on GTX 1050 graphic card.

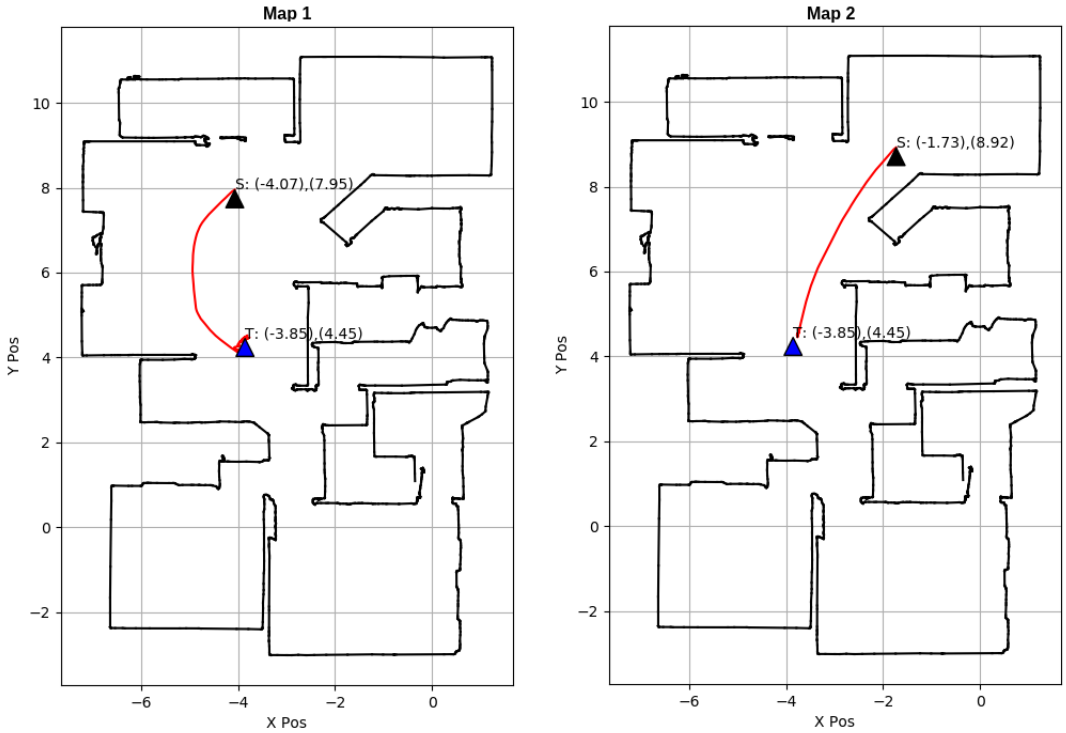


Figure 4.5 Navigation scenarios with ELM network

4.1.3.2 Comparison of Neural Network Models for Fusion Data

We experimented different kind of neural network models in order to improve navigation performance as we had mentioned before. In Figure 4.6, four parameters (cumulative reward, entropy loss, value function loss, policy surrogate function) were investigated to analyse training process of different models. The labels in the figures are named so that the first argument represents input type, the second argument represents the type of neural network model. For example, $(RGB + DEPTH + SENSOR)\_(ODE + MLP)$ represents inputs are RGB, depth and sensor data with combination ODE and MLP model.

The experiment was performed with 500 timesteps, 50 episode and 150 iterations. Hyper-parameters was set experimentally. Because, performance of training process could change according to the environment dynamics. There were lots of possible combination inputs and configurations of neural network models. In this experiment, clip-coefficient was set to 0.2 and entropy coefficient was set to 0.03. This parameters effected to training performance directly.

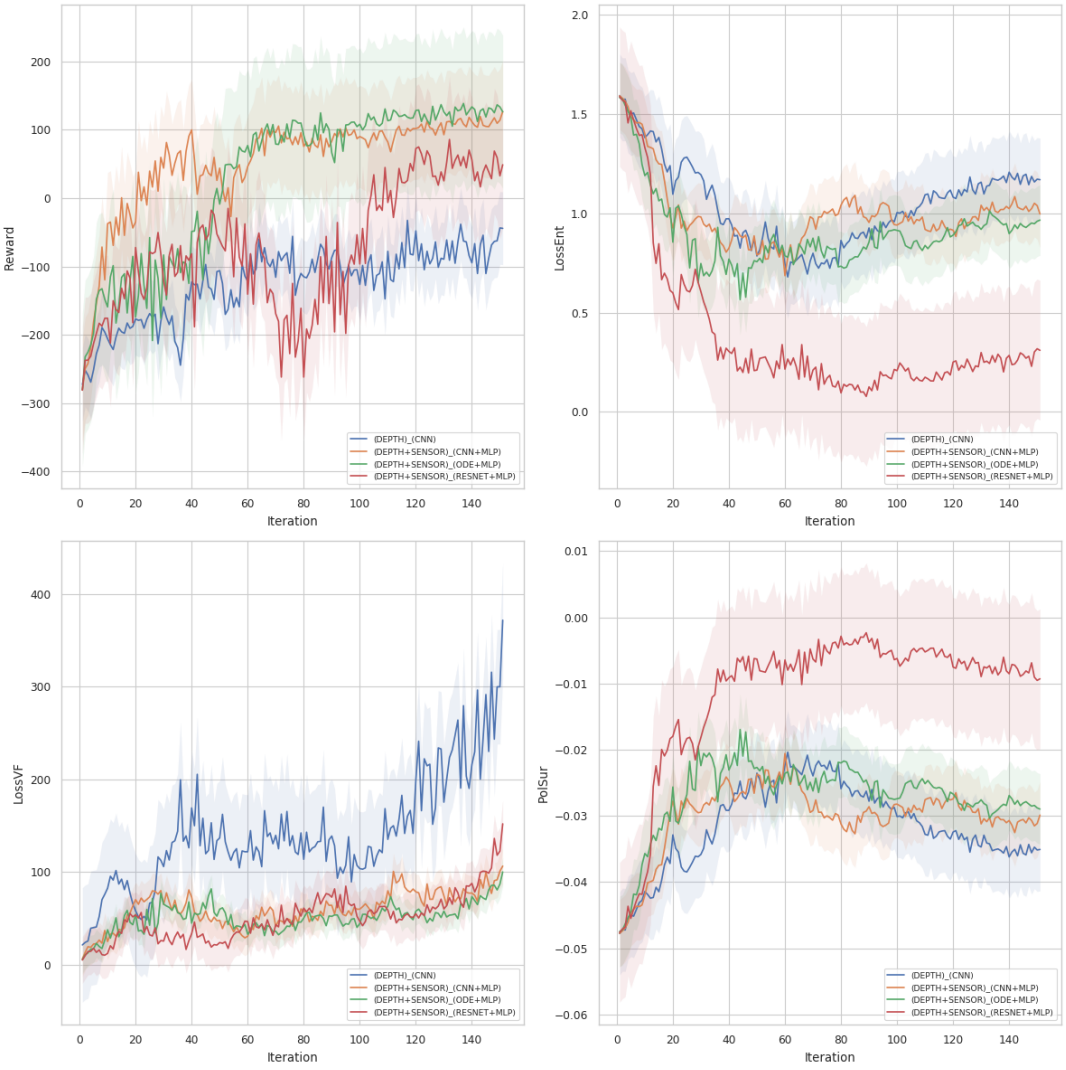


Figure 4.6 Training results of neural network models with depth images

We observed results of sensor data on MLP and ELM as it seen Table 4.2. So, we decided to use MLP model for sensor data in this environment. Combination of camera images and sensor data can be defined as fusion of data and neural network models. Results in Figure 4.6 shows that training without sensor data could not gain high cumulative rewards. As it seen, fusion data had good capability of navigation. Depth images only used in that experiment. According to the training analysis, ODE neural network model gave the best training performance with fusion data between neural network models which had been tested. Figure 4.7 indicates success rates of models during training process. ODE neural network model had also the highest rank in this benchmark.

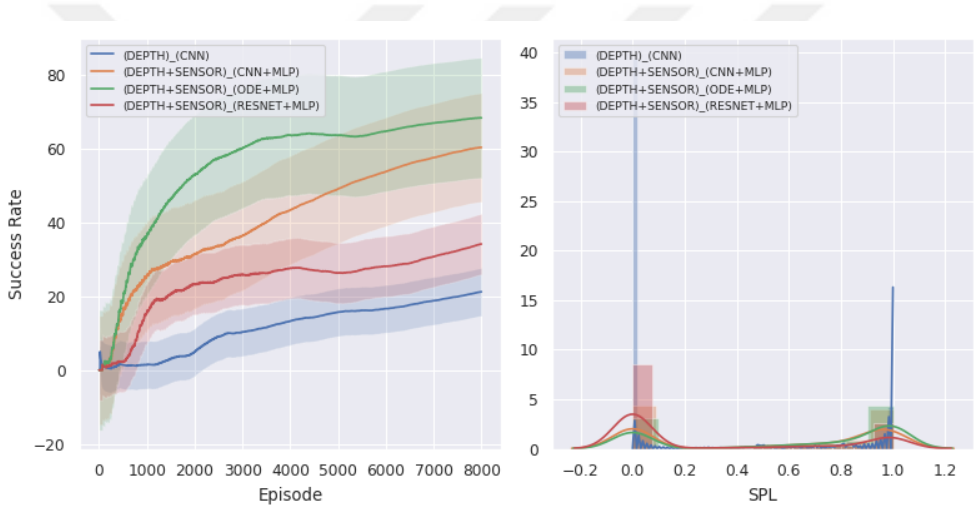


Figure 4.7 Success rates of models during training

In besides of using depth images on neural network models, RGB images were also tested with models. Figure 4.8 shows that training results. We observed that results of RGB images during traning process. ODE neural network models gave the best result in this experiment as it was at experiment with depth images. In CNN network model, RGB images couldn't be matched with correct states without sensor data. This means that, the agent faces with bad states and it has no navigation capability in order to avoid obstacles.

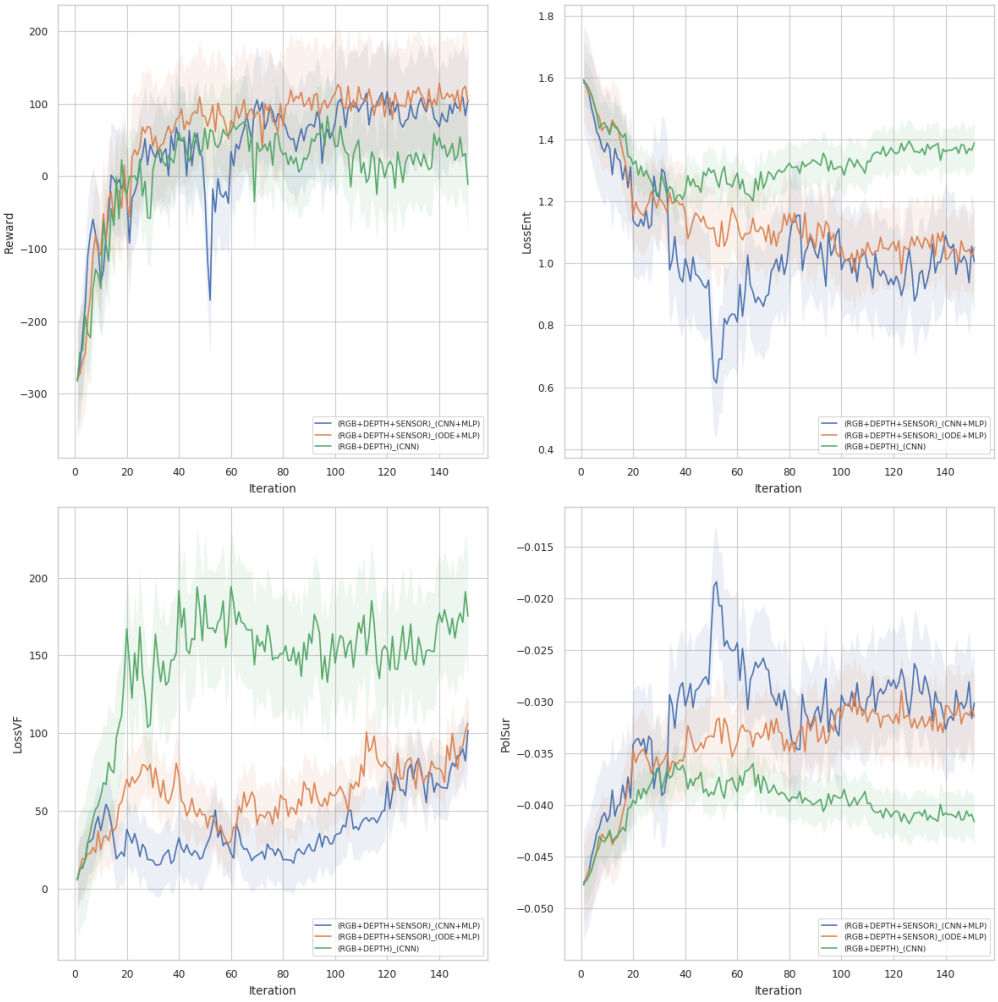


Figure 4.8 Training results of neural network Models with RGB images

Another experiment was applied with CL to understand importance of learning easy tasks to harder ones. In that experiment, As input was used fusion data with depth images and trained on combination of CNN and MLP neural network models. Figure 4.9 shows that comparison of CL and without CL success rate of method during training process. We implemented CL to all navigation experiments since we observed CL approach gave the agent the ability to navigate to target points successfully.

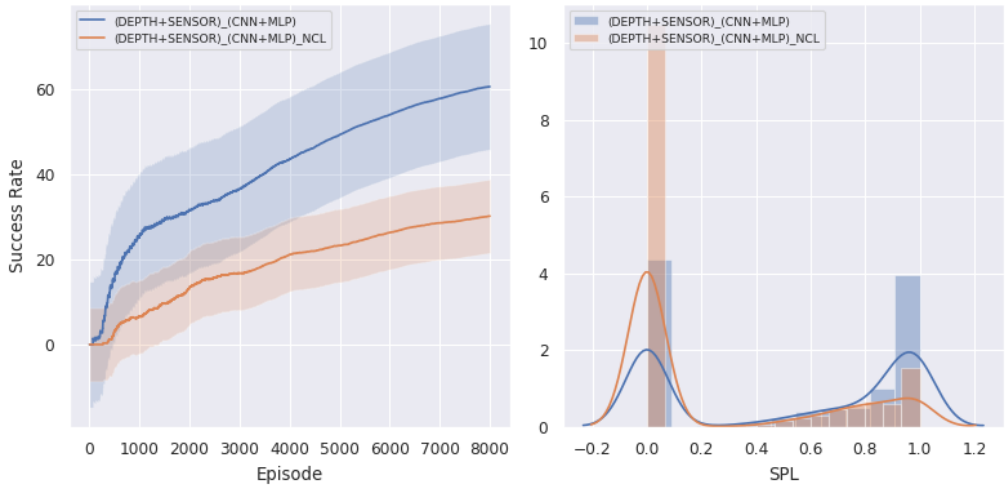


Figure 4.9 Success rates of the training with CL

Results of experiments which are given in Figure 4.6 and Figure 4.8 showed that Neural ODEs network had higher results than other network models with different input types for navigation task. Table 4.3 consists that values of average score for last 10 iteration.

Table 4.3 Numerical training results of neural network models with inputs

Inputs	**Neural Network Model**	**Average Score of Last 10 Iteration**
DEPTH	CNN	−75.384
RGB+DEPTH	CNN	36.827
DEPTH+SENSOR	CNN+MLP	110.096
DEPTH+SENSOR	ResNet+MLP	41.042
DEPTH+SENSOR	**ODE+MLP**	**128.329**
RGB+DEPTH+SENSOR	CNN+MLP	88.969
RGB+DEPTH+SENSOR	**ODE+MLP**	**112.292**

CHAPTER FIVE

CONCLUSION

DRL methods are the state-of-art algorithms that can be usable for complex navigation tasks. In this thesis, we also proposed a new approach to DRL for visual navigation in indoor environments. Using ODE neural network model in classical RL algorithm such as PPO provided effective results for navigation task in indoor environment. Visual navigation requires any sensing data to environment such as camera images. So that, we performed experiments in Gibson Environment simulation habitat which based on Pybullet physics simulator.

We chose Husky-UGV as an agent. The model of agent was integrated to simulation environment with related and applicable parameters. Especially, action space and speed control of motors were significant to reach target point in navigation. Although the continuous action space reflects the agent dynamics in real-world applications involving complex tasks, we selected the discrete action space in this study because it has less computation time and easy modeling structure.

As a next stage, we performed experiments of point-to-point navigation with using agent's proprioseptive sensor data and camera images on virtual environment. The learning algorithm was applied as policy based. To reach target, the agent improved its policy with collecting rewards from environment. One of the important points of the thesis was reward shaping. We managed that rewards as positive or negative depends on the complexity of tasks. Balance between rewards showed that the quality of navigation. Another important point was managing hyper-parameters of RL algorithm and neural network models. A good implementation of CNN model could provide more accurate results.

Early experiments showed using only one point-to-point navigation task is not enough for navigation. Because, the agent should faces various situations in same

area. If the agent has been trained in many different situations, it can gain the generalization skill. For good navigation, the agent should collects millions sample without forgetting learned states in experimental environment. We applied curriculum learning method with lots of point-to-point navigation tasks. Results indicates that curriculum learning method reduced training time to reach high success rates for navigation.

In the last stage of thesis, we applied ODE neural network model instead of traditional neural network architectures and compared different kind of neural network models. ODE solver was success to reach hidden states on the networks. We also observed a trade-off between accuracy and computation time. One of the important dedications during experiments was ELM could gave good navigation performance and reduced training time with using only sensor data. However, ELM performance can change depend on environment dynamics. The best navigation performance was obtained from ODE neural network model. It had high accuracy action selection for the states thanks to computational power of differential equations. Complex DRL problems can be solved with this model.

As future future works some improvements can be applied in order to increase success rate of navigation of agent. Training time should be increased and the agent may face with different frames amount of millions. As an another suggestion, managing reward function with different coefficients and appropriate balance of rewards in environment dynamics. Reward shaping can be implemented with using another deep learning technique such as supervised learning. Another interesting point is tuning hyper parameters of neural networks. To reduce losses of optimization different coefficients or terms can be applied in equations. Experiments with different neural network models can be performed in different virtual environments. So, ODE neural network model can be justified with different indoor scenes.

REFERENCES

Agin, B. & Demir, G. K. (2020). Derin pekiştirmeli Ögrenmede aşırı Ögrenme makineli görsel navigasyon. In 28^{th} *IEEE Sinyal İşleme ve İletişim Uygulamaları Kurultayı* 1-4.

Al-Mayyahi, A., Wang, W., Hussein, A., & Birch, P. (2017). Motion control design for unmanned ground vehicle in dynamic environment using intelligent controller. *International Journal of Intelligent Computing and Cybernetics*, *10*, 530–548.

Ali Mousavi (2020). *Off-policy estimation for infinite-horizon reinforcement learning*. Retrieved November 15, 2020 from https://ai.googleblog.com/2020/04/off-policy-estimation-for-infinite.html.

Alom, M. Z., Taha, T., Yakopcic, C., Westberg, S., Sidike, P., Nasrin, et al. (2019). A state-of-the-art survey on deep learning theory and architectures. *Electronics*, *8*, 292.

Alvarez, V. M. M., Roșca, R., & Fălcuțescu, C. G. (2020). Dynode: Neural ordinary differential equations for dynamics modeling in continuous control. *arXiv preprint, arXiv:2009.04278*. Retrieved January 5, 2021 from https://arxiv.org/abs/2009.04278.

Anderson, P., Chang, A., Chaplot, D. S., Dosovitskiy, A., Gupta, S., Koltun, V., Kosecka, J., Malik, J., Mottaghi, R., Savva, M., et al. (2018). On evaluation of embodied navigation agents. *arXiv preprint, arXiv:1807.06757*. Retrieved November 5, 2020 from https://arxiv.org/abs/1807.06757.

Arulkumaran, K., Deisenroth, M. P., Brundage, M., & Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, *34*(6), 26–38.

Beeching, E., Wolf, C., Dibangoye, J., & Simonin, O. (2019). Deep reinforcement learning on a budget: 3d control and reasoning without a supercom-

puter. *arXiv preprint, arXiv:1904.01806*. Retrieved November 12, 2020 from https://arxiv.org/abs/1904.01806.

Chen, K., de Vicente, J. P., Sepulveda, G., Xia, F., Soto, A., Vázquez, et al. (2019). A behavioral approach to visual navigation with graph localization networks. *arXiv preprint, arXiv:1903.00445*. Retrieved November 9, 2020 from https://arxiv.org/abs/1903.00445.

Chen, R. T., Rubanova, Y., Bettencourt, J., & Duvenaud, D. K. (2018). Neural ordinary differential equations. In *Advances in Neural Information Processing Systems,* 6571-6583.

Chiang, H. L., Faust, A., Fiser, M., & Francis, A. (2019). Learning navigation behaviors end-to-end with autorl. *IEEE Robotics and Automation Letters*, *4*(2), 2007–2014.

COGNUB Decision Solutions (2019). *Cognitive computing and machine learning*. Retrieved November 10, 2020, from http://www.cognub.com/index.php/cognitive-platform/.

Couman, E. (2017). Bullet Physics Library. *Open source bulletphysics.org*.

Derrick Mwiti (2020). 10 real life applications of reinforcement learning. Retrieved January 4, 2021, from https://neptune.ai/blog/reinforcement-learning-applications.

Ding, S., Zhang, J., Xu, X., & Zhang, Y. (2016). A wavelet extreme learning machine. *Neural Computing and Applications*, *27*(4), 1033–1040.

Elman, J. L. (1993). Learning and development in neural networks: the importance of starting small. *Cognition*, *48*(1), 71 – 99.

Faust, A., Ramirez, O., Fiser, M., Oslund, K., Francis, A., Davidson, J., et al. (2018). Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *IEEE International Conference on Robotics and Automation (ICRA),* 5113-5120.

Ge, S. S. & Cui, Y. J. (2000). New potential functions for mobile robot path planning. *IEEE Transactions on robotics and automation*, *16*(5), 615–620.

Ge, S. S., Xuecheng Lai, & Mamun, A. A. (2005). Boundary following and globally convergent path planning using instant goals. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *35*(2), 240–254.

Gibson, J. J. (2013). On theories for visual space perception. *Perceiving events and objects*, *11*, 179.

Grzes, M. (2017). Reward shaping in episodic reinforcement learning. In 16^{th} *Conference on Autonomous Agents and MultiAgent Systems,* 565–573.

Hacohen, G. & Weinshall, D. (2019). On the power of curriculum learning in training deep networks. In 36^{th} *International Conference on Machine Learning (ICML) 2019,* 2535-2544.

Harville, D. A. (1997). *The Moore-Penrose Inverse* (497-519). New York, NY: Springer New York.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 770-778.

Hirose, N., Xia, F., Martín-Martín, R., Sadeghian, A., & Savarese, S. (2019). Deep visual mpc-policy learning for navigation. *IEEE Robotics and Automation Letters*, *4*(4), 3184–3191.

Huang, G.-B., Zhu, Q.-Y., & Siew, C.-K. (2006). Extreme learning machine: theory and applications. *Neurocomputing*, *70*(1-3), 489–501.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., et al. (2016). Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint, arXiv:1611.05397*. Retrieved November 12, 2020, from https://arxiv.org/abs/1611.05397.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems,* 1097-1105.

Lanham, M. (2018). *Learn Unity ML-Agents–Fundamentals of Unity Machine Learning: Incorporate new powerful ML algorithms such as Deep Reinforcement Learning for games*. Birmingham, UK: Packt Publishing Ltd.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*, 436–444.

López, M. E., Bergasa, L. M., & Escudero, M. (2003). Visually augmented pomdp for indoor robot navigation. In *Applied Informatics,* 183–187.

Marom, O. & Rosman, B. (2018). Belief reward shaping in reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence,* 3762-3769.

Meng, X., Ratliff, N., Xiang, Y., & Fox, D. (2019). Neural autonomous navigation with riemannian motion policy. In *2019 International Conference on Robotics and Automation (ICRA),* 8860-8866.

Meng, X., Ratliff, N. D., Xiang, Y., & Fox, D. (2020). Scaling local control to large-scale topological navigation. *2020 IEEE International Conference on Robotics and Automation (ICRA),* 672-678.

Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A. J., Banino, A., et al. (2017). Learning to navigate in complex environments. *arXiv preprint, arXiv:1611.03673*. Retrieved November 9, 2020, from https://arxiv.org/abs/1611.03673.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., et al. (2016). Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning,* 1928-1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529–533.

Neftci, E. O. & Averbeck, B. B. (2019). Reinforcement learning in artificial and biological systems. *Nature Machine Intelligence*, *1*(3), 133–143.

Nielsen, M. A. (2015). *Neural networks and deep learning*. San Francisco, CA: Determination press.

Ocaña, M., Bergasa, L. M., Sotelo, M., & Flores, R. (2005). Indoor robot navigation using a pomdp based on wifi and ultrasound observations. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems,* 2592-2597.

Peng, X. B., Abbeel, P., Levine, S., & van de Panne, M. (2018). Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions on Graphics (TOG)*, *37*(4), 1–14.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015a). Trust region policy optimization. In *International Conference on Machine Learning,* 1889-1897.

Schulman, J., Moritz, P., Levine, S., Jordan, M., & Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*. Retrieved November 22, 2020, from https://arxiv.org/abs/1506.02438.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*. Retrieved November 18, 2020, from https://arxiv.org/abs/1707.06347.

Shi, H., Shi, L., Xu, M., & Hwang, K. (2020). End-to-end navigation strategy with deep reinforcement learning for mobile robots. *IEEE Transactions on Industrial Informatics*, 16(4), 2393–2402.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., et al. (2017). Mastering the game of go without human knowledge. *Nature*, *550*, 354–359.

Standford Vision Lab (2018). *Off-policy estimation for infinite-horizon reinforcement learning*. Retrieved November 25, 2020, from https://github.com/StanfordVL/GibsonEnv/tree/master/gibson/data.

Sutton, R. S. & Barto, A. G. (2018). *Reinforcement learning: An introduction*. Cambridge: MIT press.

Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 1-9.

Thrun, S. (1998). Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, *99*(1), 21–71.

Thrun, S. (2002). Probabilistic robotics. *Communications of the ACM*, *45*(3), 52–57.

Xia, F., Zamir, A. R., He, Z., Sax, A., Malik, J., & Savarese, S. (2018). Gibson env: Real-world perception for embodied agents. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 9068-9079.

Zhao, W., Queralta, J. P., & Westerlund, T. (2020). Sim-to-real transfer in deep reinforcement learning for robotics: a survey. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI),* 737-744.

Zhu, Y., Mottaghi, R., Kolve, E., Lim, J. J., Gupta, A., Fei-Fei, L., & Farhadi, A. (2017). Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *2017 IEEE International Conference on Robotics and Automation (ICRA),* 3357-3364.

APPENDICES

Appendix 1: Gibson Environment codes and configuration file parameters

Table A.1 Environment Configuration Parameters in YAML File

Argument name	Parameter value
envname	HuskyNavigateEnv
model_id	Euharlee
target_orn	[0, 0, 0]
target_pos	[-1,2,0.3]
initial_orn	[0, 0, 1.57]
initial_pos	[-2,-6,0.3]
robot_scale	0.5
power	2.5
control	torque
n_step	500
n_episode	50
n_iter	137
waypoint_active	true
curriculum	true
elm_active	false
test_set	false
fov	1.57
use_filler	true
display_ui	false
show_diagnostics	false
ui_num	2
ui_components	[RGB_FILLED, DEPTH]
output	[nonviz_sensor, rgb_filled, depth]
resolution	128
speed : timestep	0.01
speed : frameskip	3
mode	headless
verbose	false
enable_ui_recording	false
fast_lq_render	false

It can be reached codes of thesis framework from:

`https://github.com/Berk035/Gibson_Exercise`

Appendix 2: CNN architecture which had been used in training

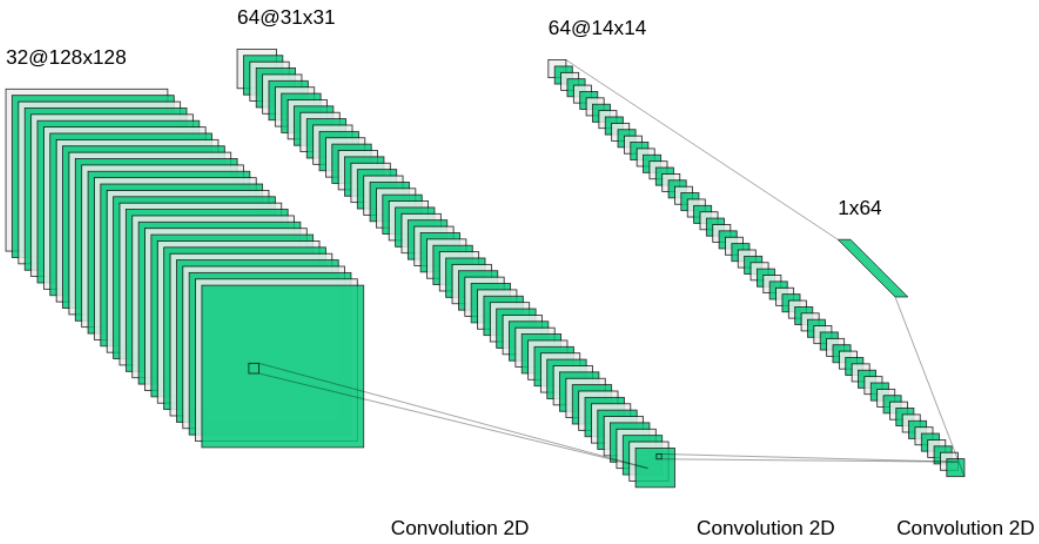
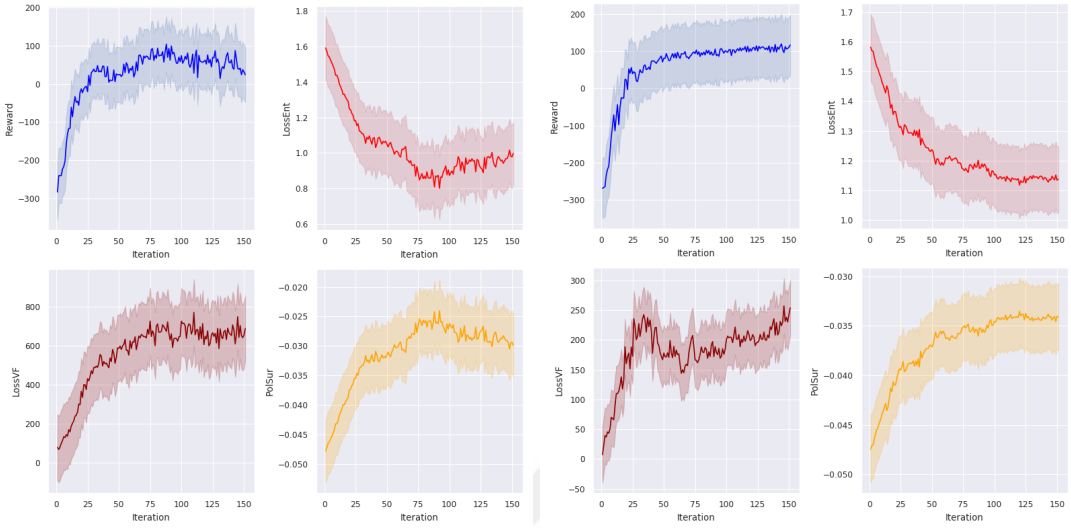


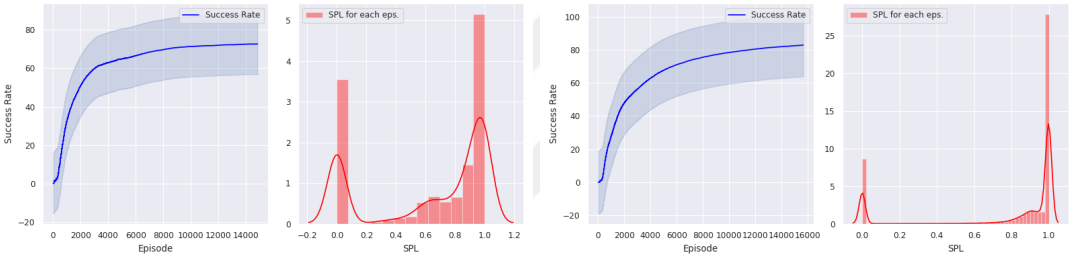
Figure A.2 CNN Architecture which had been used in training

Appendix 3: Comparison of ELM model and MLP model in ’Euharlee’ map



(a) ELM training

(b) MLP training

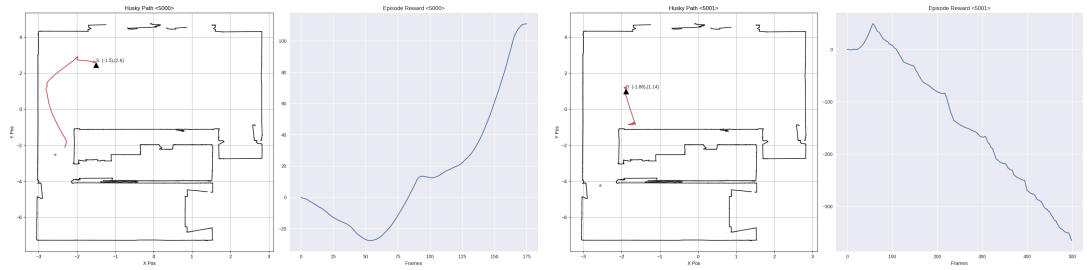


(c) Success rate of ELM in training

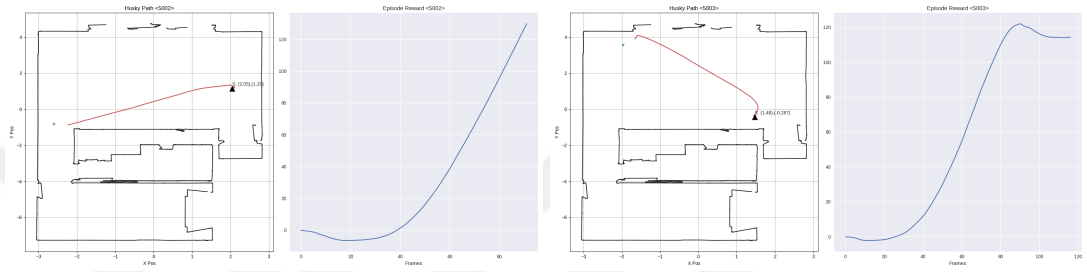
(d) Success rate of MLP in training

Figure A.3 Comparison of ELM and MLP model in ’Euharlee’ map

Appendix 4: Episode samples of neural network models with sensor data

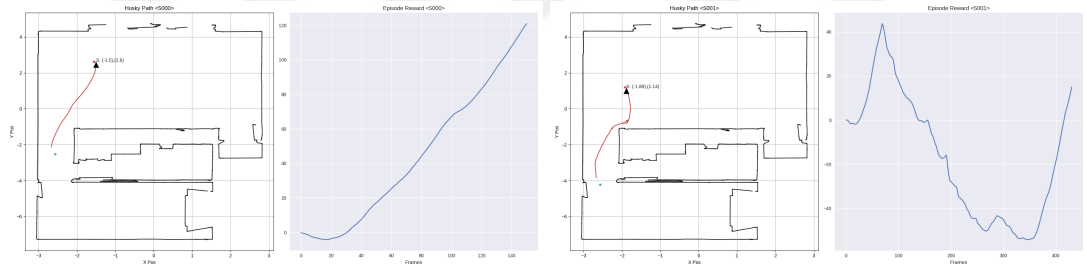


(a) Episode 5000 (b) Episode 5001

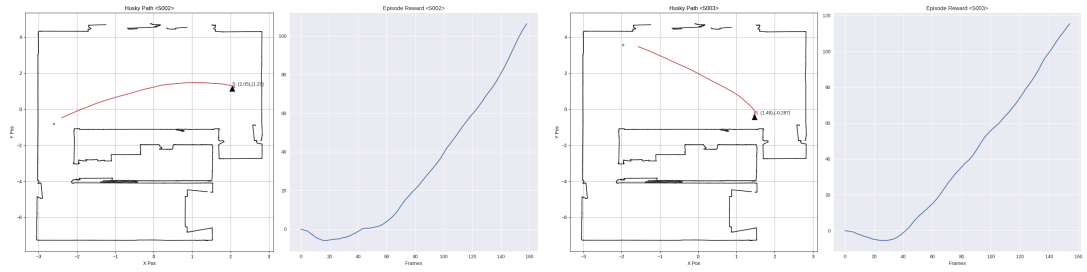


(c) Episode 5002 (d) Episode 5003

Figure A.4-1 Episode samples with SENSOR data and ELM model



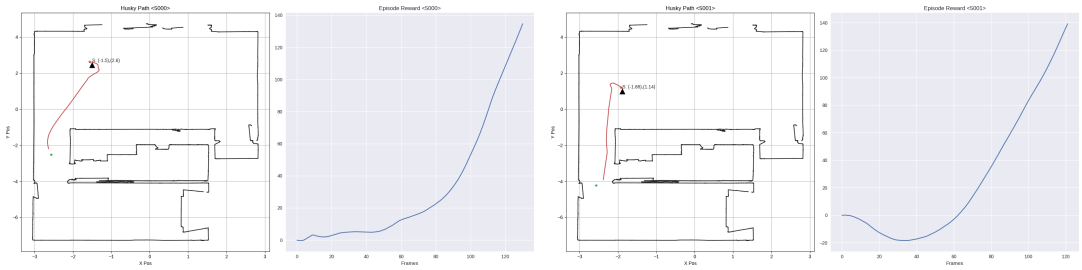
(a) Episode 5000 (b) Episode 5001



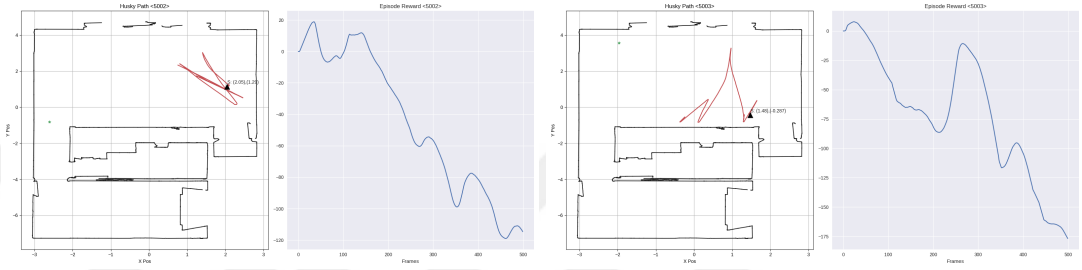
(c) Episode 5002 (d) Episode 5003

Figure A.4-2 Episode samples with SENSOR data and MLP model

Appendix 5: Episode samples of neural network models with fusion data

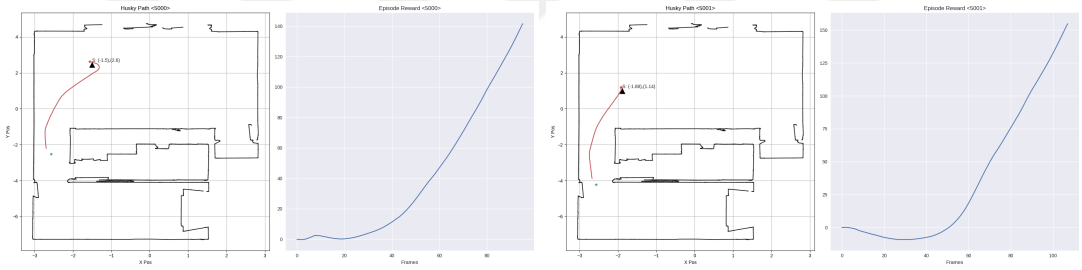


(a) Episode 5000 (b) Episode 5001

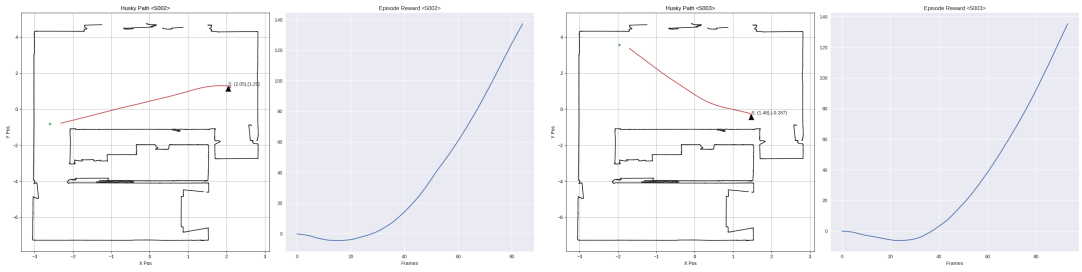


(c) Episode 5002 (d) Episode 5003

Figure A.5-1 Episode samples with DEPTH and CNN model

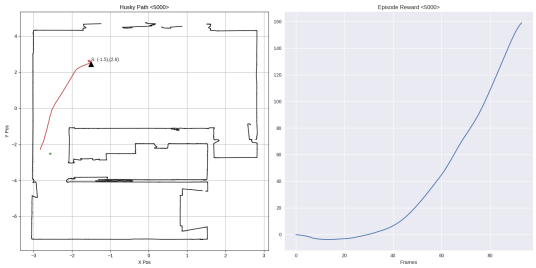


(a) Episode 5000 (b) Episode 5001

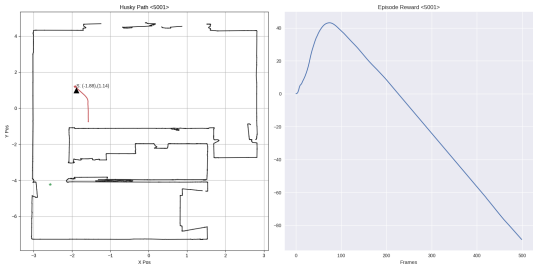


(c) Episode 5002 (d) Episode 5003

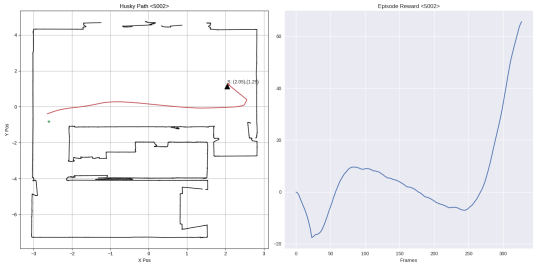
Figure A.5-2 Episode samples with DEPTH+SENSOR data and CNN+MLP model



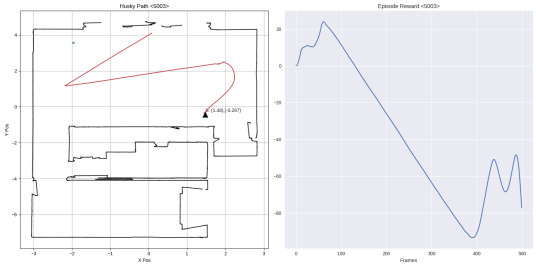
(a) Episode 5000



(b) Episode 5001

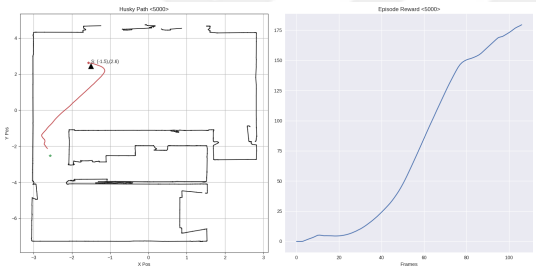


(c) Episode 5002

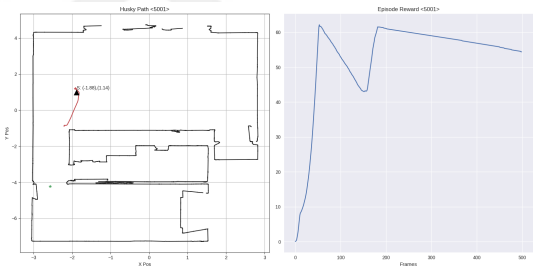


(d) Episode 5003

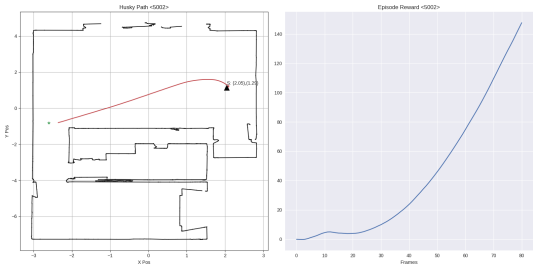
Figure A.5-3 Episode samples with DEPTH+SENSOR data and ResNet+MLP model



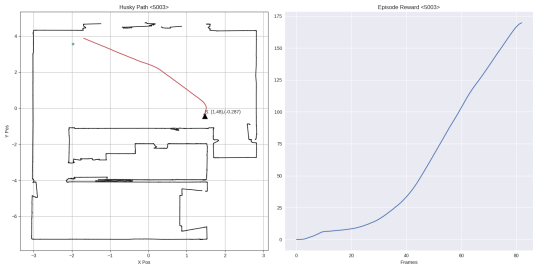
(a) Episode 5000



(b) Episode 5001

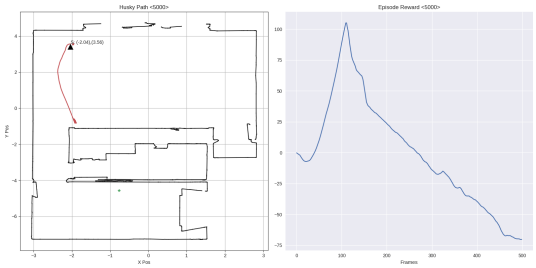


(c) Episode 5002

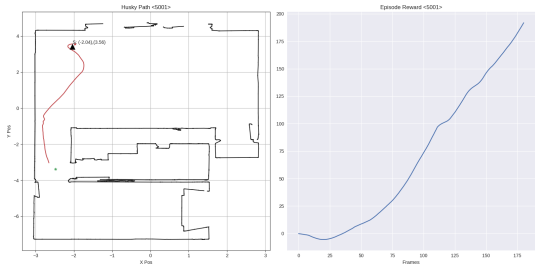


(d) Episode 5003

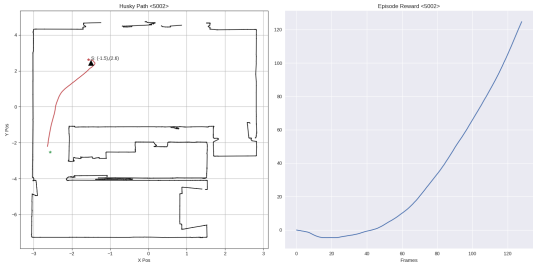
Figure A.5-4 Episode samples with DEPTH+SENSOR data and ODE+MLP model



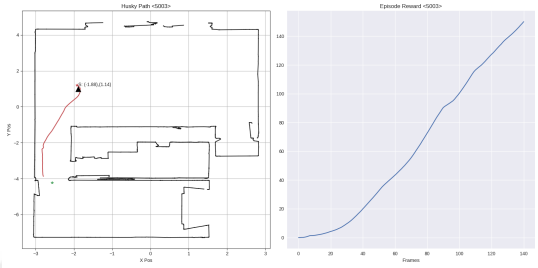
(a) Episode 5000



(b) Episode 5001

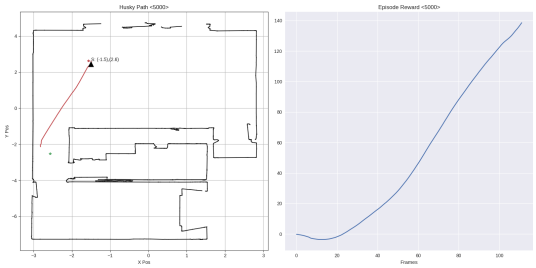


(c) Episode 5002

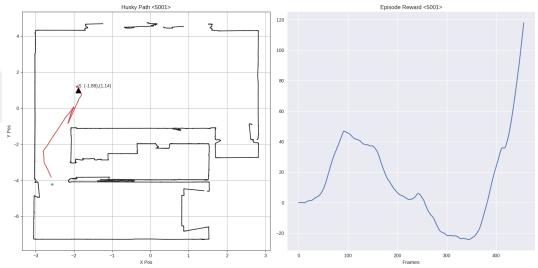


(d) Episode 5003

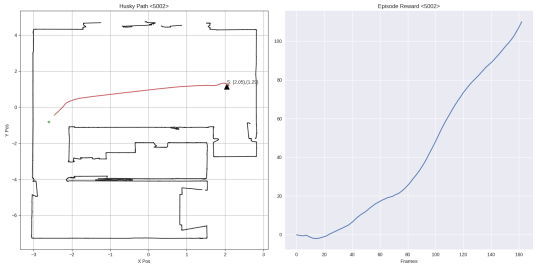
Figure A.5-5 Episode samples with RGB+DEPTH+SENSOR data and CNN model



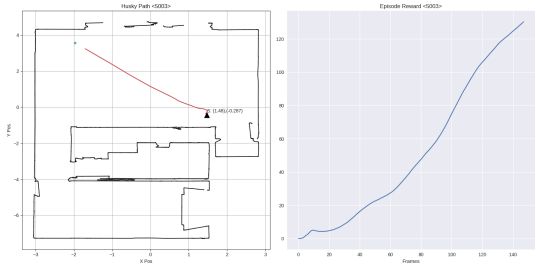
(a) Episode 5000



(b) Episode 5001

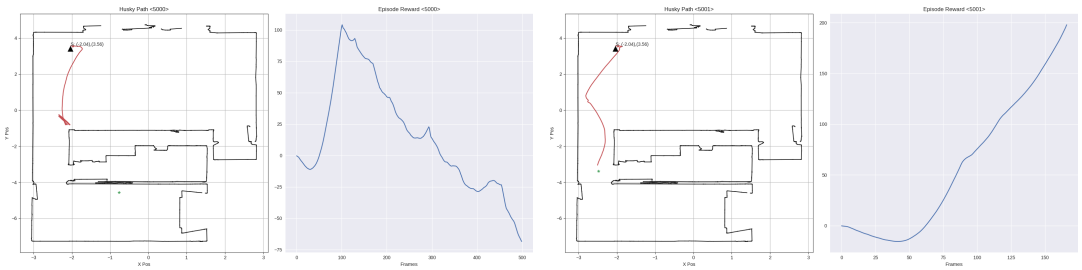


(c) Episode 5002



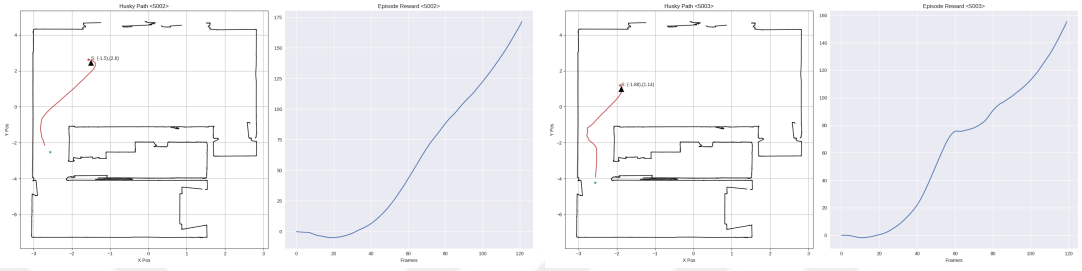
(d) Episode 5003

Figure A.5-6 Episode samples with RGB+DEPTH+SENSOR data and CNN+MLP model



(a) Episode 5000

(b) Episode 5001

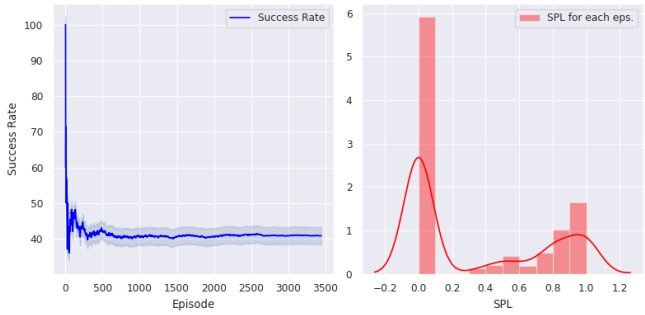


(c) Episode 5002

(d) Episode 5003

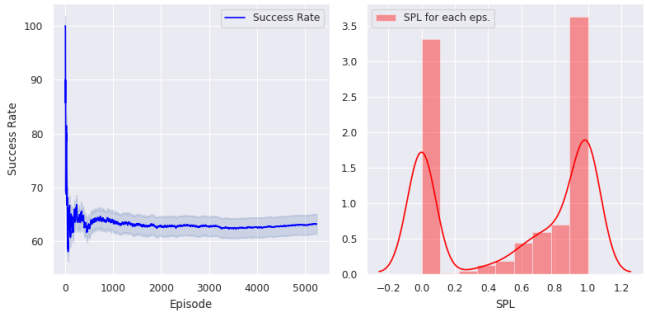
Figure A.5-7 Episode samples with RGB+DEPTH+SENSOR data and ODE+MLP model

Appendix 6: Success rates of neural network models with test dataset



(a) (DEPTH)_(CNN)

(b) (DEPTH+SENSOR)_(CNN+MLP)



(c) (DEPTH+SENSOR)_(RESNET+MLP)

(d) (DEPTH+SENSOR)_(ODE+MLP)

Figure A.6 Success rates of neural network models with test dataset