VISION BASED ROBOT NAVIGATION USING DRL METHODS

A project Report

submitted by

NIKITHA VARMA SUNCHU

EE16B152

in partial fulfilment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY &

MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

JUNE 2021

THESIS CERTIFICATE

This is to certify that the thesis titled **VISION BASED ROBOT NAVIGATION USING DRL METHODS**, submitted by **Nikitha Varma Sunchu (EE16B152)**, to the Indian Institute of Technology, Madras, for the award of the degree of **the degree of Bachelors of Technology and Master of Technology**, is a bonafide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. V. Srinivasa Chakravarthy	**Dr Mansi Sharma**
Research Guide	Research Co-Guide
Professor	Inspire Faculty
Department of Biotechnology	Department of Electrical Engineering
IIT-Madras, 600 036	IIT-Madras, 600 036

Place: Chennai

Date: 24th June 2021

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my project advisor, Prof. V. Srinivasa Chakravarthy, for giving me this oppurtuinity to be part of CNS(Computational NeuroScience) lab team. I would like to thank my mentors Azra aziz and Swetha for their immense guidance and support throughout the course of my project. I would like to express my gratitude to Dr.Mansi Sharma for supporting me being my co-guide for the project.
Finally, I want to thank my family and friends for their constant encouragement and support throughout my education and research.

ABSTRACT

KEYWORDS: Spatial Navigation, Computer Vision, Deep learning, Unity, Robot, CNN, LSTM

Recent advancements in the field of Artificial Intelligence (AI) in Robotics are impressive. AI is advancing at a breakneck pace, presenting unprecedented possibilities to improve the performance of several industries and businesses, including robotics. Highly advanced computing methodologies that imitate the way the human brain processes information are some of the AI innovations. At present, the application of mobile robots is more and more extensive, and the movement of mobile robots cannot be separated from effective navigation, especially path exploration. Aiming at navigation problems, this project proposes a method based on 3d convolution layers and LSTMs completely based on vision. This can be further extended to greater extent by using deep reinforcement learning which will also be discussed in the paper.

The data set is created by utilising the UNITY tool to create simple environments. The source code is developed in Python using the tensor flow library.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

AI	Artificial Intelligence
DRL	Deep Reinforcement Learning
LSTM	Long Term Short Memory
CNN	Convolutional Neural Network
SLAM	Simultaneous Localization and Mapping
ROS	Simultaneous Localization and Mapping
RNN	Recurrent Neural Network
MSE	Mean Squared Error
SGD	Stochastic Gradient Descent

CHAPTER 1

INTRODUCTION

The process of identifying an appropriate and safe path between a beginning and a destination point for a robot traveling between the two points is known as navigation. Different sensors have been employed for this purpose, resulting in a wide range of solutions. Since navigation techniques based on vision have become a source of numerous study contributions in the last three decades, visual navigation for mobile robots has become a source of endless research contributions.

Mobile robot navigation has typically been thought of as solving the challenge posed by these three questions:

1. Where am I?
2. Where are other places related to me?
3. How do I get to other places from here?

The answers to these questions are localization, which determines where the robot is, path-planning, which determines how to get to other locations from where the robot is, and navigation, which executes the path-planning system's commands. This method has dominated the robot navigation paradigm, with a slew of successful solutions. However, technological advancements have allowed for a more expansive view of mobile robot navigation, resulting in a solution to a "larger" problem.

This solution would answer two additional questions, which are:

1. How is this place like?
2. How is the structure of the environment I am in?

It's always inspiring to learn how the human brain works and then apply that knowledge to a problem-solving strategy. We can link Navigation to this in this situation. When we want to visit a new place, we try to understand the surroundings from the start to the finish line, and then design the most effective route in our brains automatically.

This concept is the bedrock of mobile machine navigation. In general, the two parts of the core are the perceptual environment and the planned path based on the known environment.

The emergence of simultaneous localization and mapping (SLAM) technology had a huge impact on mobile robot navigation in the early days. SLAM is a technique in which robots use sensors like vision, lasers, and odometers to build a map while learning about their surroundings.

The current SLAM problem study technique is mostly based on the placement of multi-type transmission on the robot body. The gadget employs information fusion to accomplish accurate estimation of the robot's posture and spatial modeling of the scene by estimating the motion information of the robot ontology and the feature information of the unknown environment. Visual SLAM, for example, is a system that relies on pictures as the primary source of contextual information. It has outstanding scene recognition capabilities and can gather huge and redundant texture information from the surroundings.

The main objective of a typical visual SLAM system is to estimate the camera posture and reconstruct the map using multi-view geometry theory. Some visible SLAM systems extract sparse image features first, then accomplish inter-frame estimates and closed-loop detection by matching feature points to increase data processing time. The intentionally generated sparse picture characteristics, on the other hand, now have several drawbacks. On the one hand, how to create sparse image features that best represent picture information is still a major unresolved challenge in computer vision. Sparse picture features, on the other hand, respond to variations in light.

One of the major goals of robotics and artificial intelligence is for intelligent agents to be able to navigate around an environment and carry out human-provided instructions. Visual navigation has developed as a means for such agents to learn (e.g., through deep learning) how to navigate towards a certain target object. Because the agent must learn a navigation strategy and stopping criteria conditioned on locating a specified target, the task is classified as a deep reinforcement learning problem. It's a particularly difficult challenge since the agent must learn to avoid barriers and execute actions in a complicated environment by identifying similar but visually distinct objects. Furthermore, for any beginning state, there are generally several action sequences (i.e. trajectories) that might lead to successful navigation, leave alone the trajectories that might fail. The main difficulty is figuring out how to choose the correct action at each time step to construct a path that leads to the target.

In this project, we used Deep Learning techniques using CNNs and LSTMs to learn and navigate the agent through the environment posing the navigation problem as a supervised learning problem where the inputs are camera(placed on the agent) view plus the position of the agent at that instant and output being the step size of the agent in 2d space.

The problem statement is framed as if we have an environment with obstacles and a fixed target to reach and train an agent to reach the target from different starting points in different trajectories. Can the agent learn to navigate in the environment and reach the target if left at any new starting point?

This raises few more questions like:

1. How many trajectories are enough trajectories?
2. When trained with multiple trajectories from the same start point, will the agent learn the smallest path?
3. If we train for multiple environments, can the agent perform the same when left in a new similar environment?
4. What can be done to make the agent adapt to new environments?

If the environment is initially unknown, i.e. there is no explicit map. A reinforcement learning (RL) problem may be used to formalize a visual navigation problem like this. The complexity of the agent's observation space and the fact that the real state is only partially visible from the RL formulation are two major problems. About which we are going to discuss in the future scope section.

CHAPTER 2

BACKGROUND

In recent years, Deep Reinforcement Learning (DRL) has become one of the most hotly debated areas in artificial intelligence. It combines the perception of deep learning (DL) with the decision-making abilities of reinforcement learning (RL) and uses high-dimensional perceptual input learning to control the behavior of agents directly. It proposes a novel approach to resolving robot navigation issues.

Among them, DL has gained great success in the domains of image analysis, speech recognition, natural language processing, and video classification as a key research hotspot in the field of machine learning. The main idea behind DL is to find distributed feature representations of data by combining low-level characteristics and forming abstract, readily identifiable high-level representations using multilayered network architectures and nonlinear transformations. As a result, the DL approach focuses on how people perceive and express things.

Industrial production, simulation, robot control, optimization and scheduling, and video play have all benefited from RL as a research hotspot in the field of machine learning. The primary principle behind RL is to find the best strategy for achieving a goal by maximizing the agent's cumulative reward value from the environment. As a result, the RL method emphasizes problem-solving techniques.

With the fast growth of human civilization, it is important to utilize DL to automatically learn the abstract representation of large-scale input data and use this characterization as a self-incentive RL to optimize problem-solving policy in more and more difficult real-world task tasks. As a consequence, Google's artificial intelligence research team DeepMind has created a new research hotspot in the field of artificial intelligence, called deep reinforcement learning, by combining the sensible DL with the decision-making RL.

Since then, the DeepMind team has built and implemented human expert-level agents in a variety of hard fields. These agents directly construct and learn their information.

2.1 Neural Network(NN)

The design of the neural network is based on the structure of the human brain. Just as we use our brains to identify patterns and classify different types of information, neural networks can be taught to perform the same tasks on data.

The individual layers of neural networks can also be thought of as a sort of filter that works from gross to subtle, increasing the likelihood of detecting and outputting a correct result. The human brain works similarly. Whenever we receive new information, the brain tries to compare it with known objects. The same concept is also used by a deep neural network.

The shown figure is a basic neural network. The connections of the biological neuron are modeled as weights. A positive weight reflects an excitatory connection, while negative values mean inhibitory connections. All inputs are modified by weight and summed. This activity is referred to as a linear combination. Finally, an activation function controls the amplitude of the output. For example, an acceptable range of output is usually between 0 and 1, or it could be -1 and 1.

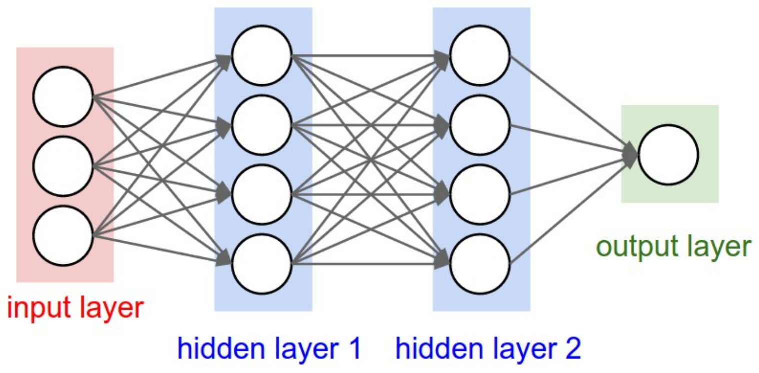

Figure 2.1: Simple Neural Network

2.2 Convolutional Neural Network(CNN)

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm that can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image, and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets can learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

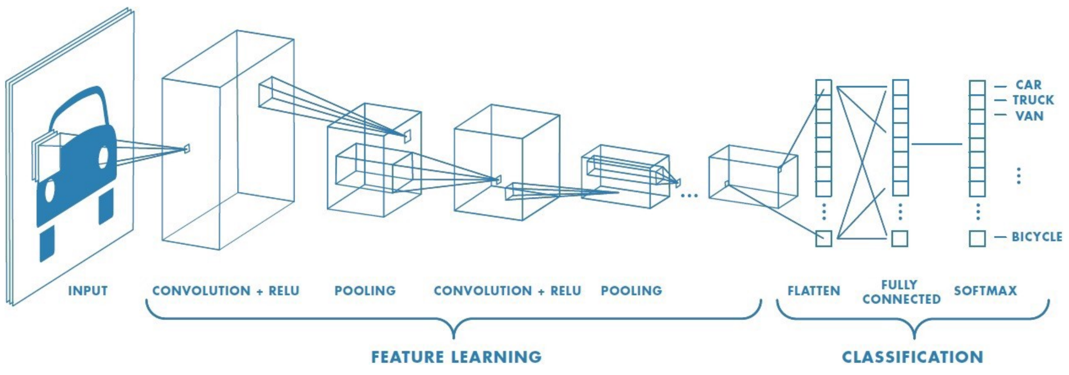

Figure 2.2: Convolutional Network

2.2.1 2D convolution Networks

This is the standard Convolution Neural Network which was first introduced in Lenet-5 architecture. Conv2D is generally used on Image data. It is called 2 dimensional CNN because the kernel slides along 2 dimensions on the data as shown in the following image.

The whole advantage of using CNN is that it can extract the spatial features from the data using its kernel, which other networks are unable to do. For example, CNN can detect edges, distribution of colors, etc in the image which makes these networks very robust in image classification and other similar data which contain spatial properties.

2.2.2 3D convolution Networks

3D CNN's are used when you want to extract features in 3 Dimensions or establish a relationship between 3 dimensions. Essentially it's the same as 2D convolutions but the kernel movement is now 3-Dimensional causing a better capture of dependencies within the 3 dimensions and a difference in output dimensions post convolution. The kernel on convolution will move in 3-Dimensions if the kernel depth is lesser than the feature map depth.

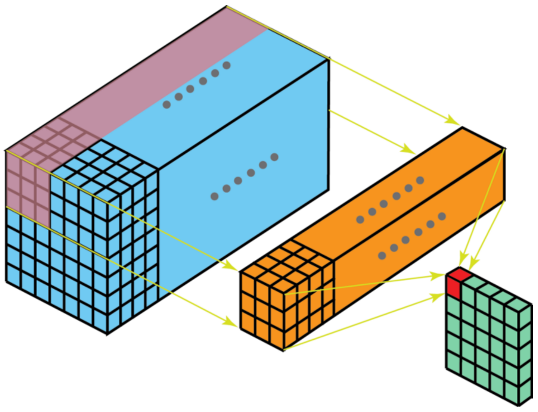

Figure 2.3: 2D convolutional Network

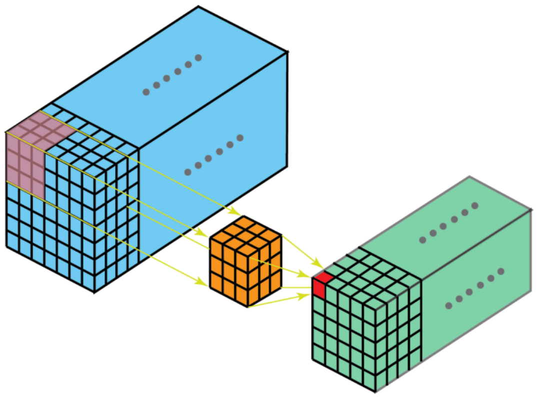

Figure 2.4: 3D convolutional Network

2.3 Recurrent Neural Networks(RNN)

A feedforward neural network with internal memory is known as a recurrent neural network. RNN is recurrent since it executes the same function for each data input, and the current input's outcome is dependent on the previous calculation. The output is duplicated and transmitted back into the recurrent network once it is created. It analyses the current input as well as the output it has learned from the prior input when making a decision.

RNNs, unlike feedforward neural networks, may process sequences of inputs using their internal state (memory). As a result, activities like unsegmented, linked handwriting recognition or speech recognition are possible. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other.

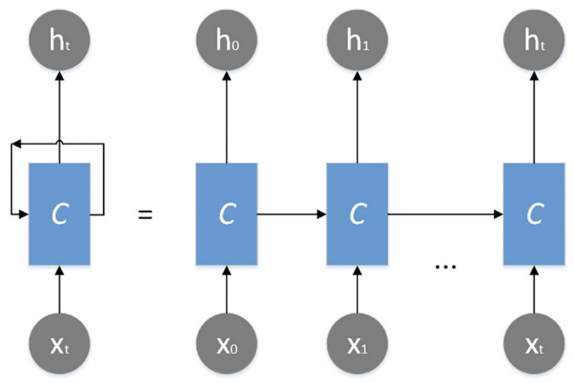

Figure 2.5: Recurrent Neural Network

2.4 Long Short Term Memory(LSTM)

Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process, and predict time series given time lags of unknown duration. It trains the model by using back-propagation. In an LSTM network, three gates are present. Input, forget and output gates.

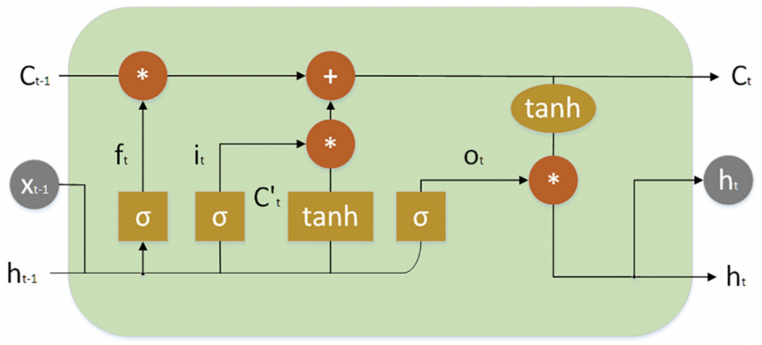

Figure 2.6: Long Short Term Memory

$$C_t = \sigma(f_t C^i + i_t C_t')$$
$$i_t = \sigma(x_t U^i + h_{t-1} W^i)$$
$$f_t = \sigma(x_t U^f + h_{t-1} W^f)$$
$$o_t = \sigma(x_t U^o + h_{t-1} W^o)$$

2.5 Convolutional LSTM

In this case, sequential images, one approach is using ConvLSTM layers. It is a Recurrent layer, just like the LSTM, but internal matrix multiplications are exchanged with convolution operations. As a result, the data that flows through the ConvLSTM cells keeps the input dimension (3D in our case) instead of being just a 1D vector with features.

2.5.1 ConvLSTM layer input

The LSTM cell input is a set of data over time, that is, a 3D tensor with shape (samples, time steps, features). The Convolution layer input is a set of images as a 4D tensor with shape (samples, channels, rows, cols). The input of a ConvLSTM is a set of images over time as a 5D tensor with shape (samples, time steps, channels, rows, cols).

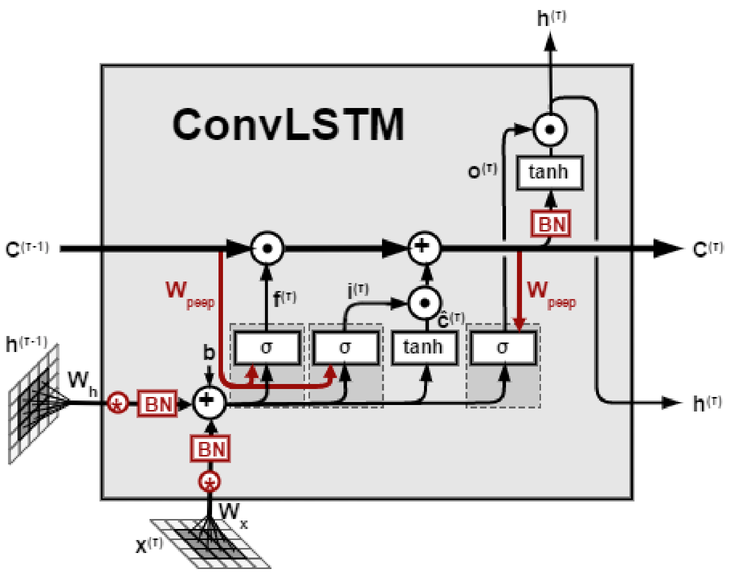

Figure 2.7: Convolutional LSTM

2.5.2 ConvLSTM layer output

The LSTM cell output depends on the return sequences attribute. When set True, the output is a sequence over time (one output for each input). In this case, the output is a

3D tensor with shape (samples, time steps, features). When return sequences is set False (the default), the output is the last value of the sequence, that is, a 2D tensor with shape (samples, features). The Convolution layer output is a set of images as a 4D tensor with shape (samples, filters, rows, cols). The ConvLSTM layer output is a combination of a Convolution and an LSTM output. Just like the LSTM, if return sequences = True, then it returns a sequence as a 5D tensor with shape (samples, time steps, filters, rows, cols). On the other hand, if return sequences = False, then it returns only the last value of the sequence as a 4D tensor with shape (samples, filters, rows, cols).

2.5.3 Other parameters

The other ConvLSTM attributes derivates from the Convolution and the LSTM layer. From the Convolution layer, the most important ones are:

- filters: The number of output filters in the convolution.
- kernel size: Specifying the height and width of the convolution window.
- padding: One of "valid" or "same".
- data format: Images format, if channel comes first ("channels first") or last ("channels last").
- activation: Activation function. Default is the linear function a(x) = x.

From the LSTM layer, the most important ones are:

- recurrent activation: Activation function to use for the recurrent step. Default is hard sigmoid (hard sigmoid).
- return sequences: Whether to return the last output in the output sequence (False) or the full sequence (True). Default is False.

CHAPTER 3

DATASET CREATION

The Dataset for our model requires a large number of camera images captured at regular intervals by the agent while navigating around the environment. The model is trained using these images. Such datasets are either scarce or very expensive to buy from internet sources. On the other hand, we wanted to have control over the environment so that we could guide the agent in the newly constructed environment not just during the training phase, but also during the testing phase. Our early trials were performed with ROS and Gazebo software, which is commonly utilized in most research publications to create their environments. Issues with ROS were tough to overcome because the versions are constantly updated and changing, and the gazebo environment was too dark to extract appropriate pictures.

The code for agent movement and image capture was also done relatively easily since Unity is linked with Visual Studio by default, and the language used was C# (C sharp), which was close to CPP and hence a little easier than ROS CPP. The Unity 2109 version was used on the Windows 7 desktop.

3.1 Environment and Agent

The environment is designed to follow the rules of physics as they exist in the actual world. It's designed to look like a closed room, with four walls and a non-slip floor. On top of that, there is a light source. It even has a shadow effect to make it look like the actual world. The scene is painted in a variety of hues, much as in the real world. A red cube serves as the target, with rectangular obstacles strewn about at random. Space is 10x10, with each wall painted a different color. As a first step, we created the simplest environment we could think of to test our hypothesis regarding the problem statement.

For navigating in the environment, a simple cylindrical agent is developed. The cylindrical agent has a camera attached to its head that functions as our eye view camera,

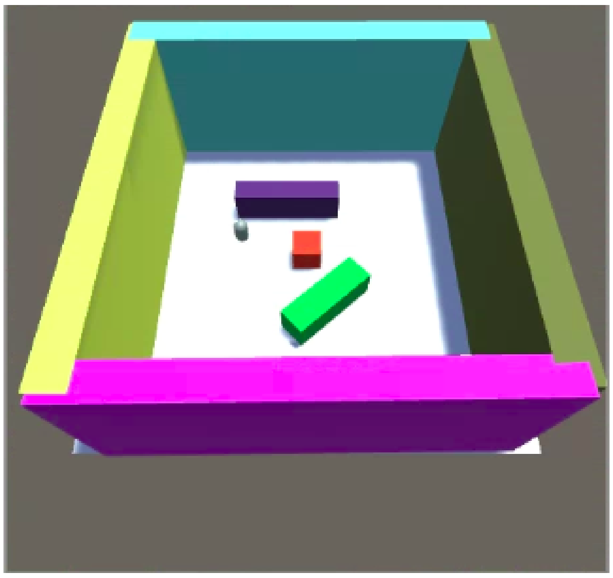

Figure 3.1: Environment

extracting images of the environment as viewed by the agent. The agent is given all of the physical attributes to closely resemble a real-world agent. Elastic collisions occur between the agent and immovable walls and objects. A mesh collider on the agent helps it bounce back a little from kinematic collisions due to recoil. The agent's Gravity Properties are also used. User controls(Keyboard and mouse) cause the agent to move in translation and rotation. The camera head that is connected to the body does not have any independent action.

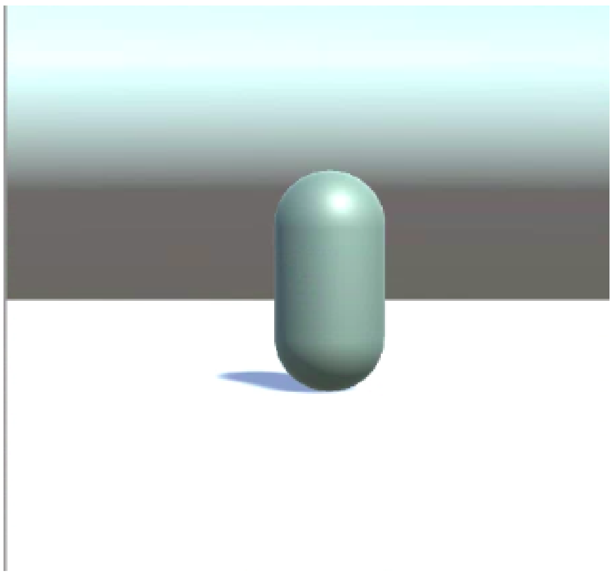

Figure 3.2: Agent

3.2 Dataset Format

Both the keyboard and the mouse may be used to control the agent's navigation through the environment. The agent goes along several paths that are drawn with the mouse(We chose mouse as it is easy to draw curved paths). The trajectories are designed in such a manner that they can begin at any place in the surroundings and go to the target. To train the agent, we create a dataset that will be used for both training and testing. The various dataset components are explained below.

3.2.1 Camera Images

This collection of camera images can be likened to a real-world scenario seen via human eyes. Because the camera is connected to the agent's head, the camera images depict the scene as it appears to the agent. This information is essential for the agent to comprehend the environment and recognize the many objects that it contains. The images are captured at regular intervals along with the agent's movement along any given trajectory.

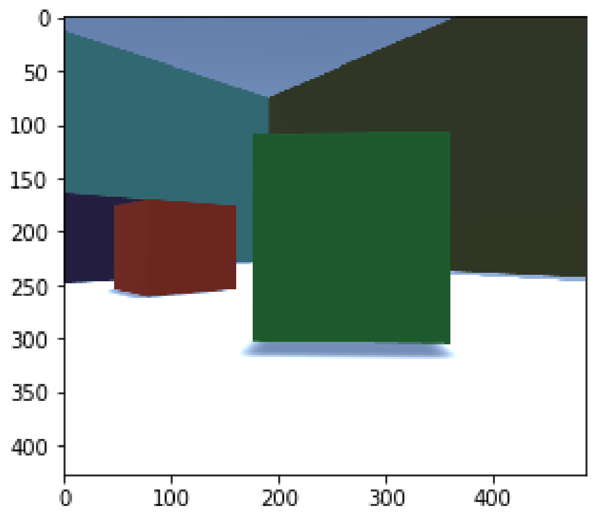

Figure 3.3: Camera view image

3.2.2 Heat Maps

We are attempting map-based navigation utilising DL approaches without using RL methods at first. So, instead of using a map, we're using a heat map to enter the agent's position. A heat map is a black-and-white grid representation that depicts the agent's position in the environment by making the block where it is situated white. We provide the position of the agent in the form of an image to the neural network.

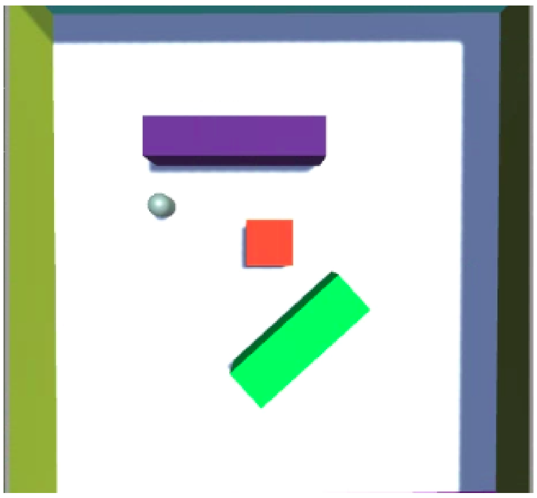
Figure 3.4: Top View

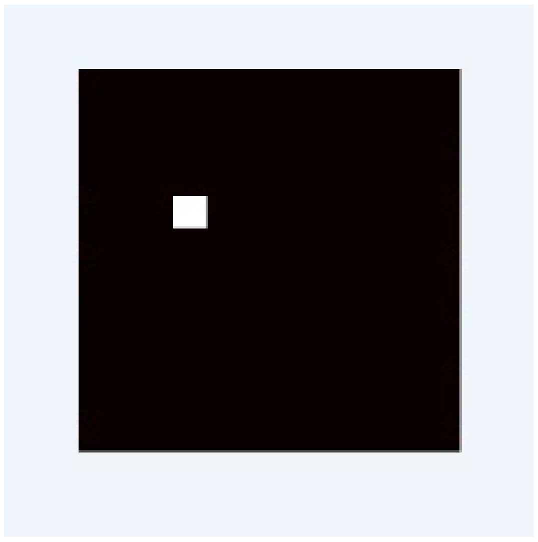
Figure 3.5: Heat Map

3.2.3 TSV

The other input for the neural network is the class of the image. The number of classes depend on number of targets present in the environment. In the simplest case as considered now with single target the classes are two: Class 0 : The Target is not in the vicinity of the agent. Vicinity radius considered here is small r (can be varied).
Class 1 : The Target is in the vicinity of the agent. Vicinity radius considered here is small r (can be varied). The target is present in the camera image.
Similarly for n targets n+1 classes are present.

3.2.4 Ground Truth/ output

The output or ground truth to train or test the model in our case is the step size taken by the agent in every next instant given present state of agent. The step size is considered in both directions x and y thus this also gives the direction change in the agent. Ground truth is calculated by taking the difference between position of the next instant and present instant. We want the model to learn the step it has to take at every instant to reach the target finally.

dx = x(t+1) - x(t)

dy = y(t+1) - y(t)

3.3 Data Preprocessing

3.3.1 Image Resizing

Once we've cleaned up the data, we'll have image data to work with, which can be used to train the model. However, we may speed up the model by changing the input data, or in this example, resizing the image size. This would still have all of the necessary functionality for us. We shrink the original image from 450*450 pixels to 40*40 pixels. In addition, the heat maps are just black and white representations. This would result in a three-channel picture, which isn't required in this case. We downsize this into a 40*40 picture by converting it to a gray scale image with only one channel. Reducing the image size also has a regularising impact, which is more realistic in real-world settings where sensors provide noisy data.

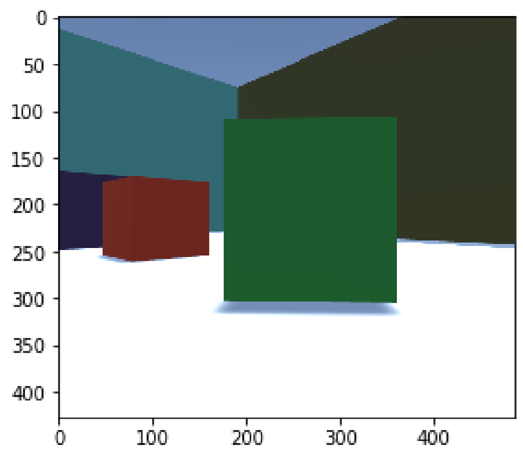

Figure 3.6: Actual Image

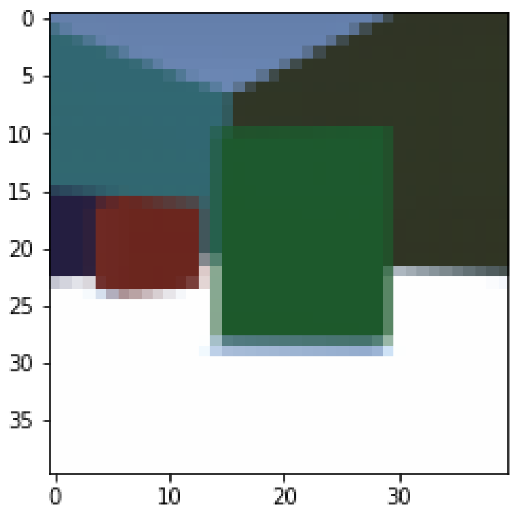

Figure 3.7: Resized Camera Image

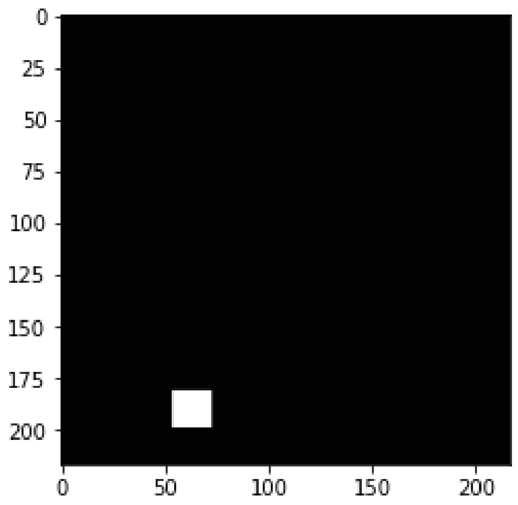

Figure 3.8: Original Heat Map

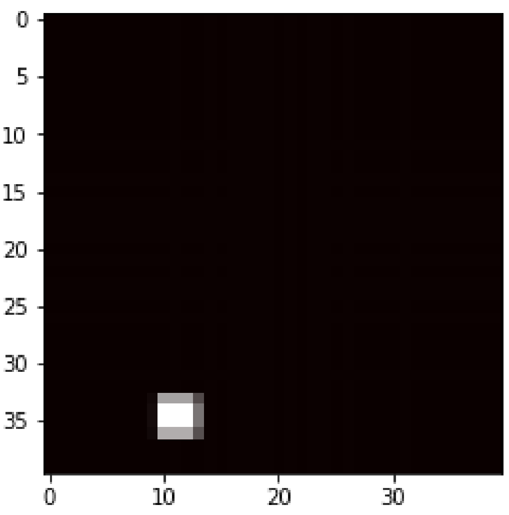

Figure 3.9: Resized Heat Map

3.3.2 Batching the data

Batch size is the number of training examples in one forward/backward pass. The higher the batch size, the more memory space you'll need. Batch size > 1 helps in memory management and also effective & efficient training of the model. Here we batch size as 10. This helps while mini batch gradient descent.

3.3.3 Window sizing for Convolutional LSTM

To incorporate the memory into the neural network we use ConvLstm layers. As the data we send is sequential data and agent also has to learn the relation between consequent frames we want the network to have memory. In ConvLstm layer inputs the second dimension out of 5 dimensions represent the time sequence that is number of previous frames it has to relate to with the present frame in order to derive the relational information. Here we are taking 5 as the Window size for ConvLstm.

3.3.4 One hot encoding for TSV

As the Class is categorical variable we have to one hot encode the variable before sending it into the network. Here we are binary coding the TSV.

3.4 Dataset collection

The Datasets are collected in two ways to test hypothesis.The two types of dataset collections are explained below:

3.4.1 Same start point

As to see if agent can learn to reach the target from a given start point, how many trajectories does it need? and also What path will the agent finally take if left at that point in case of testing? Will be shortest path or the one of the path from train set? To answer these questions we have randomly chosen a start point and taken 5 trajectories from the same point to target with minimal changes within trajectories. This is the first kind of dataset we created.

3.4.2 Multiple start points

The second type of dataset is created using different start points and multiple trajectories from each start point to the target avoiding the obstacles. This should answer if the agent can learn the environment and given any random start point, will it reach the target without hitting obstacles? Will it perform well even when the start point is not in the train dataset?

CHAPTER 4

ARCHITECTURE AND TRAINING

4.1 Architecture

ConvLSTM2D layers, Conv3d layers, and Denselayers are the main components of the model's architecture. The batch normalisation and max pooling actions are carried out as needed. The first branch converts the camera input into LSTM layers, while the second branch handles the heatmaps. They are concatenated into a single layer after batch normalisation. They are then transferred to the LSTM layer and subsequently into a sequence of Convolution3d layers once the concatenation is completed. They are then flattened down to form a dense layer.

We choose "mse" as our loss since we're looking for a distance whose formula is similar to mse's. The sgd optimizer is set at a learning rate of 0.01 since it is a convenient low and stable rate. When we are trapped at a local minimum, a higher learning rate does not always offer us a global minimum, but a lower learning rate does not converge the model adequately. Additionally, the SGD optimizer is favoured over the Adam optimizer, which appears to overfit the model even with a small number of layers and a suitable learning rate.

We adjusted the weights in the model that are learned from the first dataset after we finished training it on one dataset. Now we load the second dataset and retrain the model with the learnt weights using this dataset. As they gain experience, the weights improve. This method is continued until all of the training datasets have been finished.

4.2 Training

After deciding on what all components to use, the only thing left is hyper parameter tuning. This was done by trial and error method by looking at loss curves at the end of each epoch and also total training. The final model looks something like this :

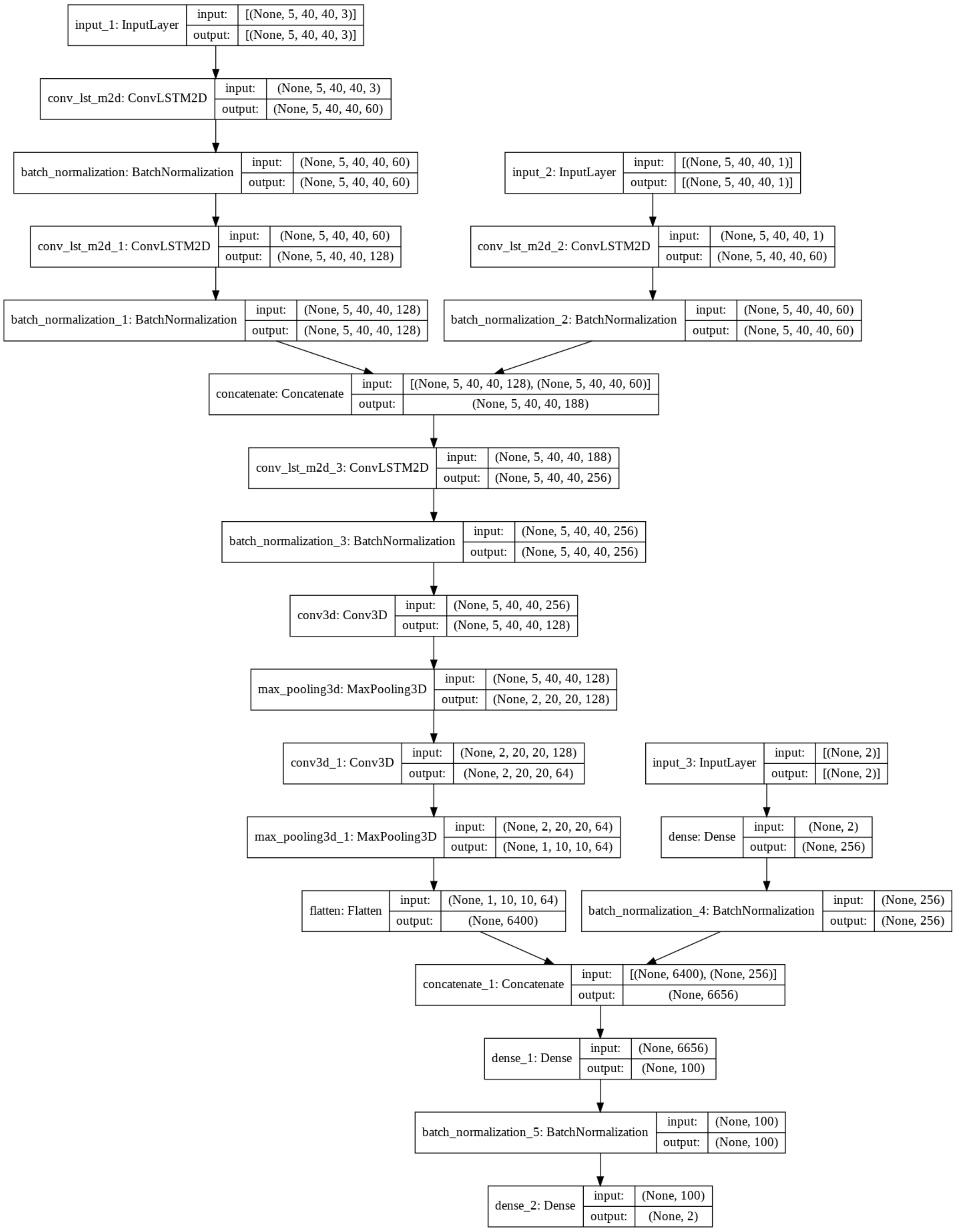
input_1: InputLayer | input: [(None, 5, 40, 40, 3)] | output: [(None, 5, 40, 40, 3)]
conv_lst_m2d: ConvLSTM2D | input: (None, 5, 40, 40, 3) | output: (None, 5, 40, 40, 60)
batch_normalization: BatchNormalization | input: (None, 5, 40, 40, 60) | output: (None, 5, 40, 40, 60)
conv_lst_m2d_1: ConvLSTM2D | input: (None, 5, 40, 40, 60) | output: (None, 5, 40, 40, 128)
batch_normalization_1: BatchNormalization | input: (None, 5, 40, 40, 128) | output: (None, 5, 40, 40, 128)
input_2: InputLayer | input: [(None, 5, 40, 40, 1)] | output: [(None, 5, 40, 40, 1)]
conv_lst_m2d_2: ConvLSTM2D | input: (None, 5, 40, 40, 1) | output: (None, 5, 40, 40, 60)
batch_normalization_2: BatchNormalization | input: (None, 5, 40, 40, 60) | output: (None, 5, 40, 40, 60)
concatenate: Concatenate | input: [(None, 5, 40, 40, 128), (None, 5, 40, 40, 60)] | output: (None, 5, 40, 40, 188)
conv_lst_m2d_3: ConvLSTM2D | input: (None, 5, 40, 40, 188) | output: (None, 5, 40, 40, 256)
batch_normalization_3: BatchNormalization | input: (None, 5, 40, 40, 256) | output: (None, 5, 40, 40, 256)
conv3d: Conv3D | input: (None, 5, 40, 40, 256) | output: (None, 5, 40, 40, 128)
max_pooling3d: MaxPooling3D | input: (None, 5, 40, 40, 128) | output: (None, 2, 20, 20, 128)
conv3d_1: Conv3D | input: (None, 2, 20, 20, 128) | output: (None, 2, 20, 20, 64)
input_3: InputLayer | input: [(None, 2)] | output: [(None, 2)]
max_pooling3d_1: MaxPooling3D | input: (None, 2, 20, 20, 64) | output: (None, 1, 10, 10, 64)
dense: Dense | input: (None, 2) | output: (None, 256)
flatten: Flatten | input: (None, 1, 10, 10, 64) | output: (None, 6400)
batch_normalization_4: BatchNormalization | input: (None, 256) | output: (None, 256)
concatenate_1: Concatenate | input: [(None, 6400), (None, 256)] | output: (None, 6656)
dense_1: Dense | input: (None, 6656) | output: (None, 100)
batch_normalization_5: BatchNormalization | input: (None, 100) | output: (None, 100)
dense_2: Dense | input: (None, 100) | output: (None, 2)

Figure 4.1: Final Model

CHAPTER 5

RESULTS

The train and test results are given here. The loss and accuracy curves are shown below.Initially in order to check if the model is learning well with the paths in environment we made a sample dataset with 5 paths. 4 for training and 1 for testing. These 5 datasets start from the same point and end at the same point with a little variations in the paths taken.

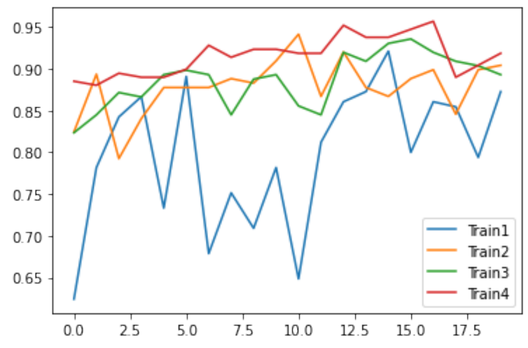

Figure 5.1: Accuracy

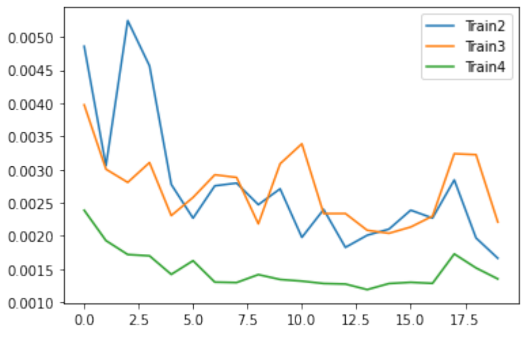

Figure 5.2: Loss

After training for 20 epochs we could see that the model training accuracy was increasing with paths and loss was becoming more and more stable. This model gave us a test accuracy of 0.8707 which proved that model was not only stable but also learning properly.

The model learns the path from a single start point gradually, Each new trajectory gives new information for as long as there is no more new information is available for the agent to learn. So each new train trajectory improves the agents performance and then saturates after some point as the learning is at max further than that the agent might over fit so we stop and set hyper parameters there.

CHAPTER 6

FUTURE SCOPE

More complex environments and multiple targets can be used to make the agent adapt to new environments.

RL techniques can be used like Actor Critic models to make problem statement into unsupervised learning from supervised learning problem. Thus the agent can learn and adapt to new environments.

The head rotation can be a key point to implement, By this the problem statement implicitly changes to have sub goals like the agent should not only learn how the target is but also where is that target object most likely be. Like if the target was a paper weight it would be mostly on table, desk or on pile of papers. So making the head to rotate in z axis and x,y axis we can imitate the human navigation process more realistically.

REFERENCES

[1] Vision-based Navigation Using Deep Reinforcement Learning by Jonáš Kulhánek, Erik Derner, Tim de Bruin, Robert Babuška

[2] Virtual-to-real Deep Reinforcement Learning: Continuous Control of Mobile Robots for Mapless Navigation by Lei Tai1, Giuseppe Paolo3, Ming Liu2

[3] A novel mobile robot navigation method based on deep reinforcement learning by Hao Quan, Yansheng Li, Yi Zhang

[4] Mobile Robot Navigation in Indoor Environments: Geometric, Topological, and Semantic Navigation By Ramón Barber, Jonathan Crespo, Clara Gómez, Alejandra C. Hernámdez and Marina Galli

[5] The Neuroscience of Spatial Navigation and the Relationship to Artificial Intelligence by Edgar Bermudez Contreras, Benjamin J Clark, Aaron Wilber