Robot Navigation With RL

Robot Navigation
with

MapBased

Deep Reinforcement Learning

01 Layered Costmaps

Is created from multi-sensor data and stacked as layers from each source where the top is the master costmap

02 Reinforcement Learning

A convolutional neural network is trained as a function approximator for the action-value function of the RL problem

03 Robot Navigation

The robot is trained in the Gazebo simulator on a gradually increasing set of difficult tasks, using the CNN with the costmaps as input

Costmaps

1. One Layered Costmap

A monolithic costmap cannot describe correctly a real dynamic environment

2. Limited Information

Updating the costmap is problematic as sensor data might be in conflict with other sources

3. Fixed Update Areas

Lack of semantic information make it difficult to determine the area that needs to be updated

4. Semantically Fixed Interpretation

The monolithic costmap only allows one type of data(e.g., is or isn't there an obstacle). It's not possible to combine both probabilistic or other kind of values in the costmap

5. Many Contexts and Sources

While the original monolithic costmap only contains binary data, i.e. if there is or isn't an obstacle, the layered costmaps can contain varied types of data in its layers and only make the cutoff decision when aggregating into the master costmap

D. V. Lu, D. Hershberger, and W. D. Smart, "Layered costmaps for context-sensitive navigation," in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2014, pp. 709–715

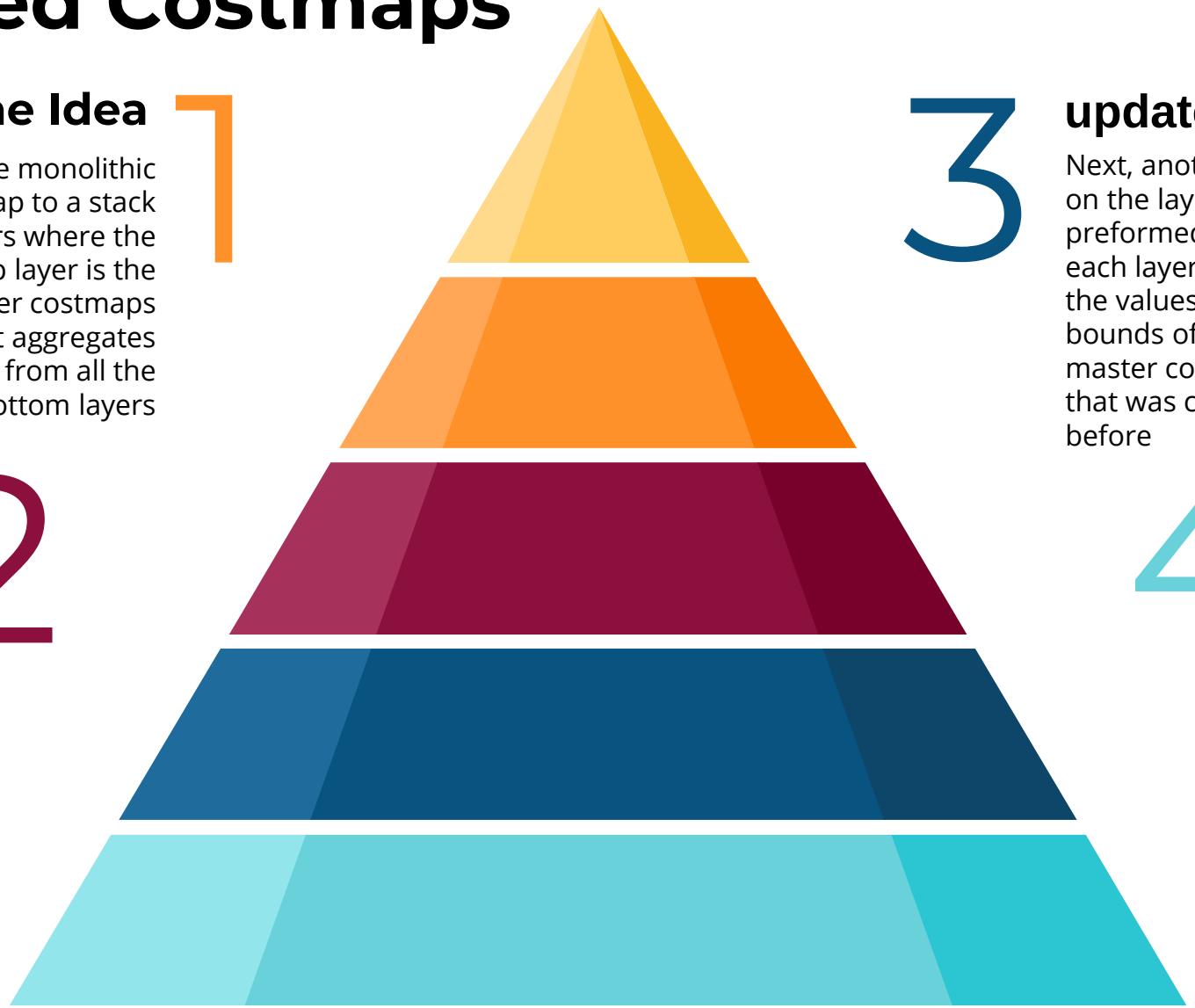
The Layered Costmaps

1 The Idea

Split the monolithic costmap to a stack of layers where the top layer is the master costmaps that aggregates data from all the bottom layers

2 UpdateBounds

The update process starts with the bottom layer which updates the region of the costmap that should be updated. Each layer adds upon the bounds of the layer beneath it until we reach the master costmap



3 updateValues

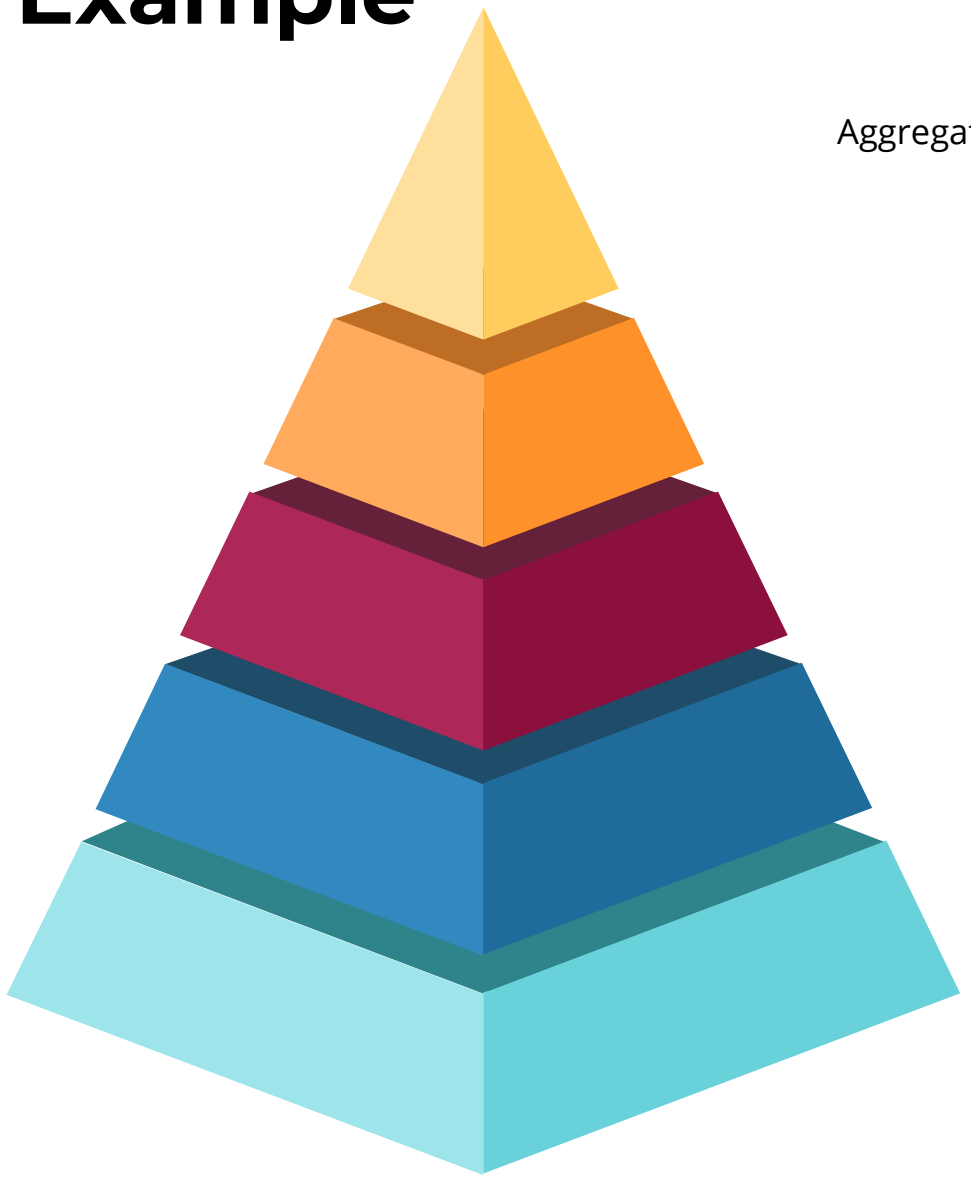
Next, another pass on the layers is preformed in which each layers updates the values in the bounds of the master costmaps that was created before

4 Separation of concerns

It is important to note that each layer cannot overwrite data written by other layers. That means that if an obstacle from the original map layer is not present, it will still appear as though it is still there

D. V. Lu, D. Hershberger, and W. D. Smart, “Layered costmaps for context-sensitive navigation,” in Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2014, pp. 709–715

An Example



Aggregation of all other layers **Master Costmap**

Inserts buffer zone around each lethal obstacle **Inflation Layer**

Layer that is created using lasers and RGB-D cameras **Obstacle Layer**

Preferrable moving regions, such as to prevent access to kitchens, etc.. **Regions Layer**

Map that was generated using SLAM algorithm a priori **Static Layer**

Updating The Costmaps

1

The layered costmap before the update

2

In the first pass, the updateBounds is called on each layer. Each one expands the region

3

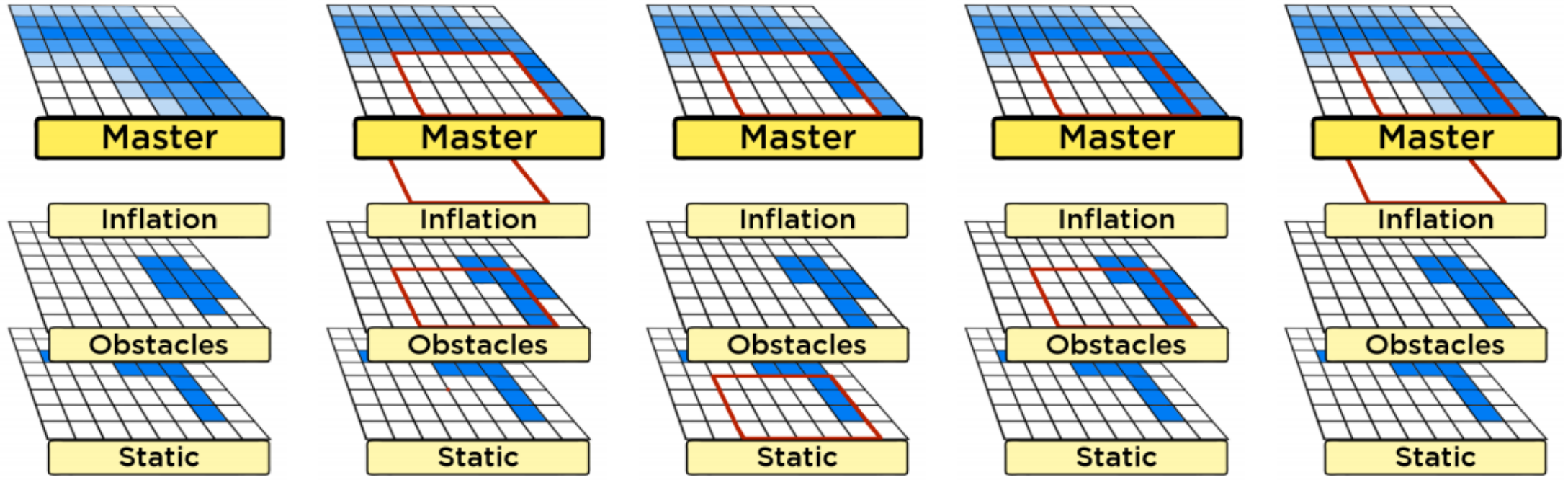
In the second pass, the updateValues is called in the aggregated region on the bottom layer

4

Next, updateValues on the next layer

5

Finally, the values are aggregated into the master costmap



(a) Initial Costmap Values (b) UpdateBounds (c) UpdateValues: Static (d) UpdateValues: Obstacles (e) UpdateValues: Inflation

Reinforcement Learning

1. Definition

The RL problem consists of an agent which learns from experience and an environment in which he interacts with

2. Markov Decision Process

We can describe the environment using an MDP. This is a mathematical framework to describe the interaction between the agent and the environment

3. Observations, Rewards & Actions

In each step, the agent takes an action, gets a reward and an observation from the environment. Next, the agent takes another action.

4. Our Goal

The goal is to maximize our reward by learning a policy. We can achieve that by approximating an action value function that will tell us how much reward we can get if we are at a certain state and take a particular action in which we continue acting using our policy.

Q-Learning

Definition 01

Our goal is to find action-value function Q*. Q* is defined as the max of the expectation for the total sum of reward, if we are at state **s**, taking action **a** and then following policy **π**.

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right]$$

The Bellman Equation 02

We can write Q* using an identity known as the Bellman Equation in which we can split the equation into the reward we get from taking the action and then taking the max of all the actions we could take next using Q*.

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') \middle| s,a\right]$$

Iterative Update 03

We can learn Q* using an algorithm called iterative update from actual experience in which each iteration we update each value using values we calculated in the previous iteration from rewards we got while interacting with the environment.

Dueling Double DQN

A function approximator for the action-value function

V. Mnih, K. Kavukcuoglu, D. Silver et al., “Playing atari with deep reinforcement learning,” 2013, https://arxiv.org/abs/ 1312.5602

H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI), 2016.

Wang, Ziyu, et al. “Dueling network architectures for deep reinforcement learning.” arXiv preprint arXiv:1511.06581 (2015)

***This is caused because we are doing a max operation over actions, and since our environment is a MDP and the amount of reward we get is stochastic, our Q- action value is only an estimation and thus we will get over estimation with high probability**

A

DQN

Deep Q-Learning attempts to approximate the action-value function using deep learning

B

Double DQN

Double Deep Q-Learning attempts to prevent ***over estimation** of Q-action values by decoupling between the selection of actions and evaluating them while preforming iterative updates

C

Dueling DQN

Suggests splitting the action-value function into two parts- the value function and the advantage function

V. Mnih, K. Kavukcuoglu, D. Silver et al., “Playing atari with deep reinforcement learning,” 2013, https://arxiv.org/abs/ 1312.5602

H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI), 2016.

Wang, Ziyu, et al. “Dueling network architectures for deep reinforcement learning.” arXiv preprint arXiv:1511.06581 (2015)

Deep Q-Learning

Plus, Prioritized Experience Replay

1. Problem

Finding the true Q* is hard if not impossible as the state space might be infinite or experiencing every state if problematic

2. Solution

By using a function approximator, we can approximate states and actions that we haven't seen before

3. Method

By using a convolutional neural network, we can build such approximator with great accuracy

4. Experience Replay

As a neural network assumes the training data is i.i.d, and the data we feed into is highly corelated due to the way we experience the environment, by saving previous experience and feeding it again into the training process we dis-corelate our training data

5. Prioritized Experience Replay

An improvement on experience replay in which we increase the replay probability of experience tuples that have a high expected learning progress

V. Mnih, K. Kavukcuoglu, D. Silver et al., "Playing atari with deep reinforcement learning," 2013, https://arxiv.org/abs/ 1312.5602

Double DQN

1. Problem

Original DQN algorithm overestimates future rewards as it uses for selecting next action and evaluating next action-value the same neural network

2. Solution

Create another network called the target network which will be feezed for certain number of timestamps, the old network is called the online network

3. Method

When updating the network, we will pick the best action using the online network but evaluate the next state using the target network

Evaluation of next action-value

$$Q(s,a;\theta) = r + \gamma Q(s', argmax_{a'} Q(s', a'; \theta); \theta')$$

Selection Of Next Action Off-Policy

Online Network Weights

Target Network Weights

H. Van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double q-learning,” in Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI), 2016.

Dueling DQN

1. Problem

Coupling between states and actions, which prevents learning state values regardless of the action the agent might take

$$Q(s,a) = V(s) + A(s,a)$$

2. Solution

Split the action-value function into a value function and an action advantage function

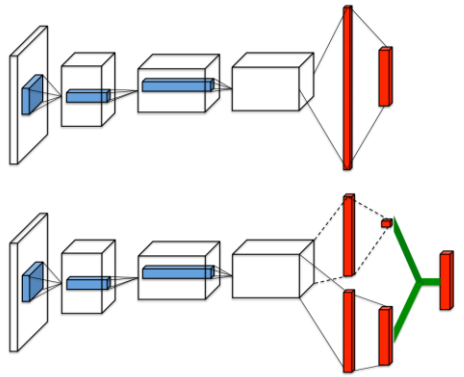
3. Aggregation

Aggregating the final Q value is not straightforward as you won't be able to extract the functions using the result of the action-value function. However, by changing the equation and forcing the highest Q-value to be equal to the value V, making the highest value in the advantage function be zero and all other values negative, could allow the network to calculate them in the back propagation step.

$$Q(s,a) = V(s) + (A(s,a) - max_{a' \in |\mathcal{A}|} A(s,a))$$

3. Method

Add another hidden layer that separates between the state value function and the action advantage function



Wang, Ziyu, et al. “Dueling network architectures for deep reinforcement learning.” arXiv preprint arXiv:1511.06581 (2015)

Robot Navigation

1. SLAM

simultaneous localization and mapping (SLAM) loads the environment map and establishes the robot location and velocity using the robot sensors

2. Path Planner

A series of local goal points are calculated from the current position to the target location

3. Layered Costmaps

A stack of layered costmaps is generated from fusion of several sensor inputs. The basic layers are the static map, the obstacle layer and master costmap

4. Dueling Double DQN

A CNN based Dueling Double DQN is trained as the function approximator for the action value function. The inputs of the network are the layered costmaps, the relative position of the next local goal and the current velocity of the robot. The outputs of the network is one-hot encoding of 28 discrete actions to preform which translates to linear and angular velocities

5. Curriculum Learning

Finally, the robot is trained in a series of tasks with increasing order of difficulty in the Gazebo simulator and in the real world

G. Chen *et al.*, "Robot Navigation with Map-Based Deep Reinforcement Learning," *2020 IEEE International Conference on Networking, Sensing and Control (ICNSC)*, 2020, pp. 1-6, doi: 10.1109/ICNSC48988.2020.9238090.

SLAM & Path Planning

The **SLAM (Simultaneous Localization and Mapping)** is a technique to draw a map by estimating current location in an arbitrary space. Next, a path plan can be created.

Mapping

A map is created using the sensors of the robot(e.g., lasers)

Localizing

Next, the robot location can be established in the map(using a particle filter)

Planning

Finally, using the map and location, a path be calculated to any point in the map.

Double Dueling DQN

Inputs

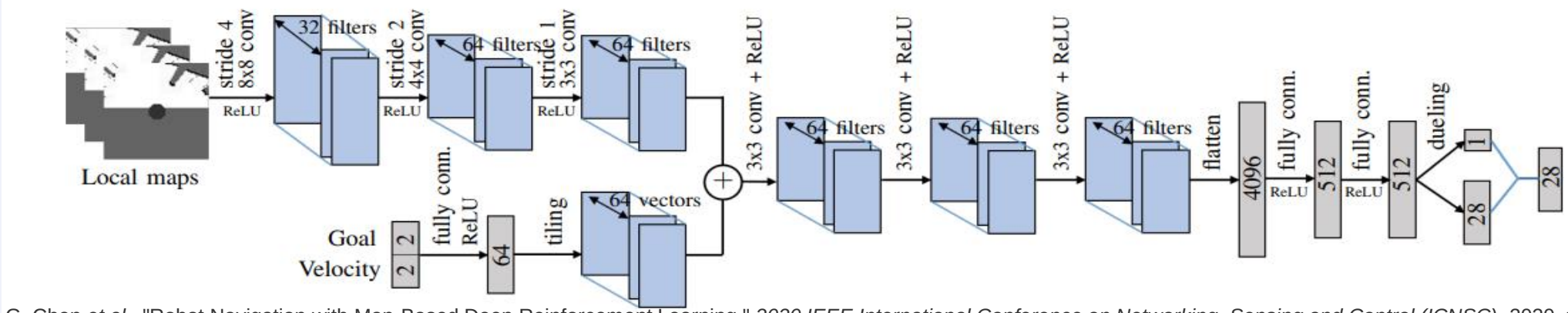
The network accepts three local cost maps, the relative goal position and robot velocity

Netowork

A CNN-based deep convolutional neural network with the dueling hidden layer

Output

A one-hot encoded 28 possible actions that translates to linear and angular velociites



G. Chen *et al.*, "Robot Navigation with Map-Based Deep Reinforcement Learning," *2020 IEEE International Conference on Networking, Sensing and Control (ICNSC)*, 2020, pp. 1-6, doi: 10.1109/ICNSC48988.2020.9238090.

Navigation

Sensors

Multi sensor data is fusioned into the costmap generator

SLAM

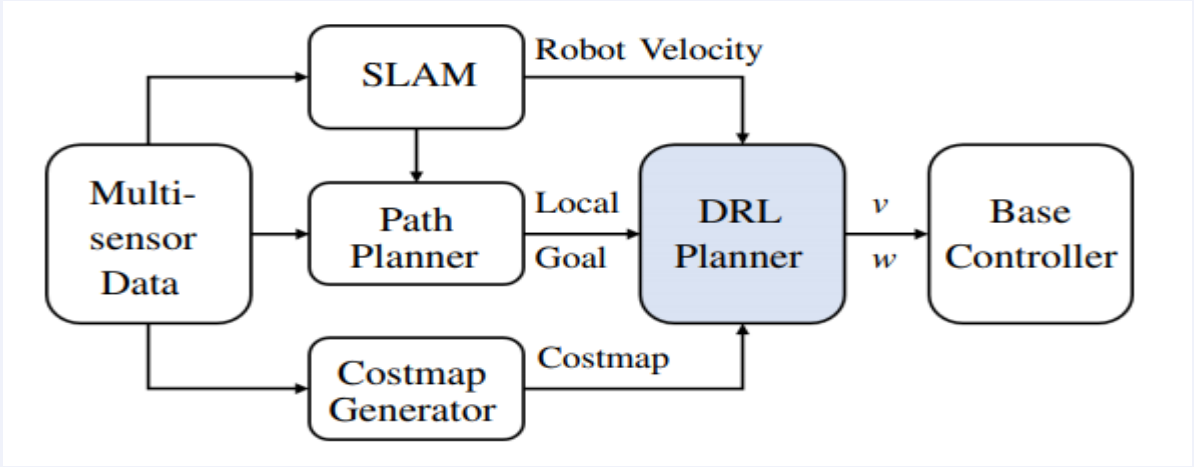
The location and velocity of the robot is established

Path Planner

A path is established to the goal

DRL

The Deep Reinforcement network establishes a collision free linear and angular velocities



G. Chen *et al*., "Robot Navigation with Map-Based Deep Reinforcement Learning," *2020 IEEE International Conference on Networking, Sensing and Control (ICNSC)*, 2020, pp. 1-6, doi: 10.1109/ICNSC48988.2020.9238090.

Curriculum Learning

From Easy To Hard

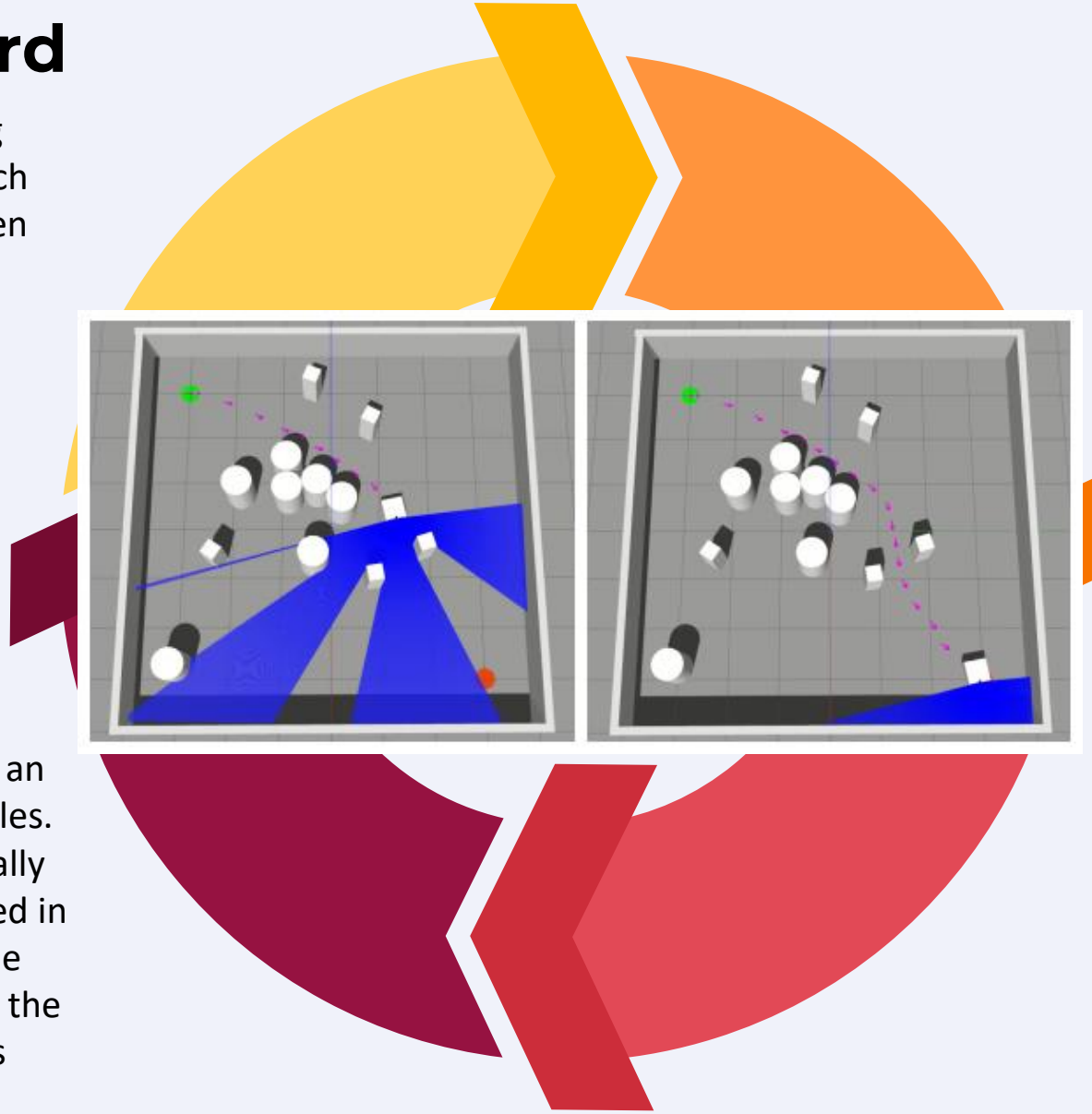
Curriculum learning is a learning strategy in machine learning, which starts from easy instances and then gradually handles harder ones

TABLE I
SYSTEM PARAMETERS

Parameter	Value
learning rate	5×10^{-4}
discount factor	0.99
replay buffer size	2×10^{5}
minibatch size	1024
image size	60×60
episode length	300
initial exploration	1
final exploration	0.1

Gazebo

Gazebo simulator is used to build an environment with multiple obstacles. As the training progresses, gradually the number of obstacles is increased in the environment. In addition, the distance from the starting point to the target point gradually increases

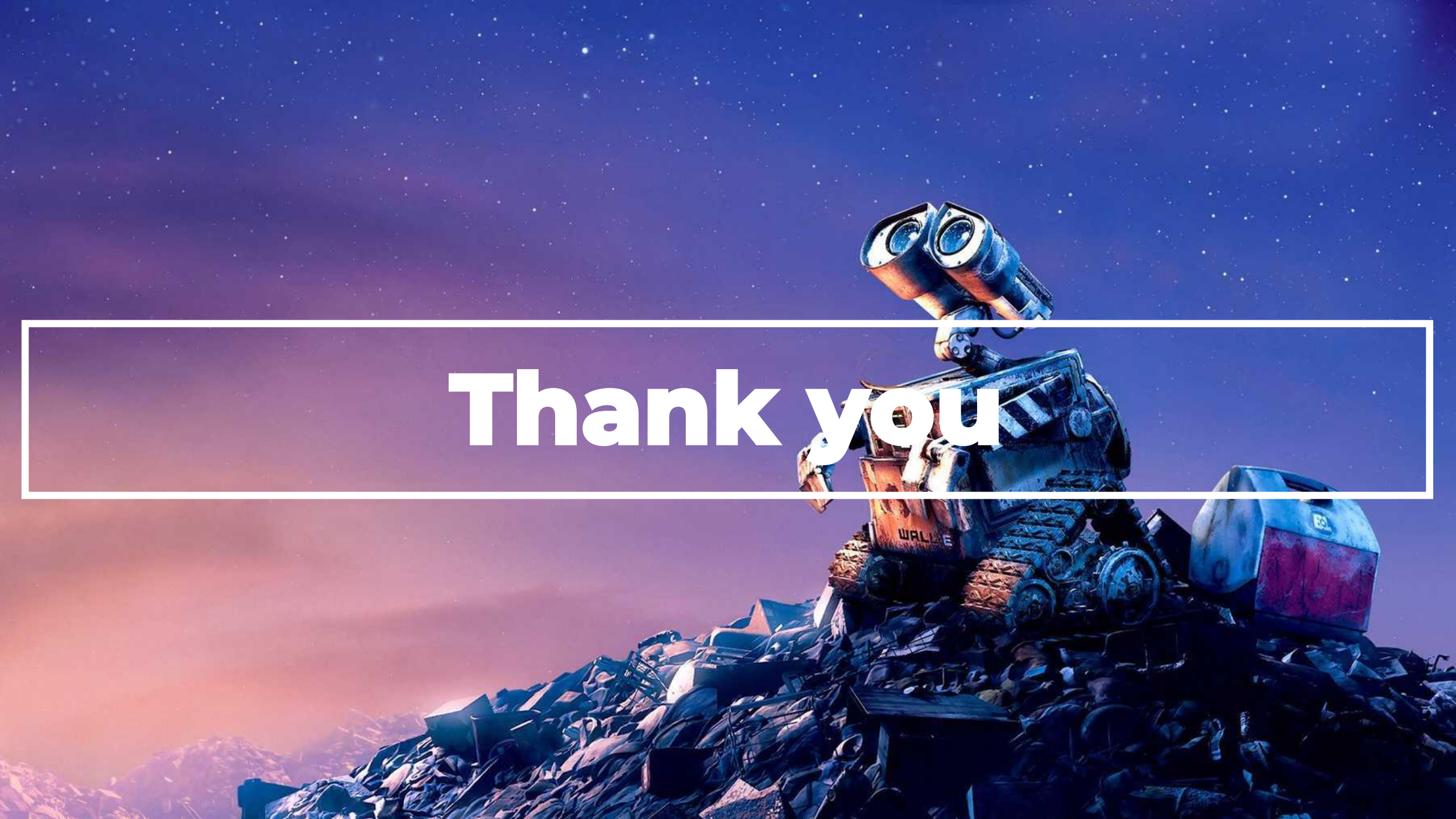


Random

The position of each obstacle and the start and end points of the robot are random during all training episodes

Real World

The robot is also trained on a series of tasks in the real world with static and dynamic obstacles such as people


Thank you