Motion Planning in Dynamic Environment

Valerii Serpiva, Artem Lykov, Mikhail Litvinov, Mikhail Konenkov

Step-by-step implementation of Q-learning and DQN

Motion Planning in Dynamic Environment

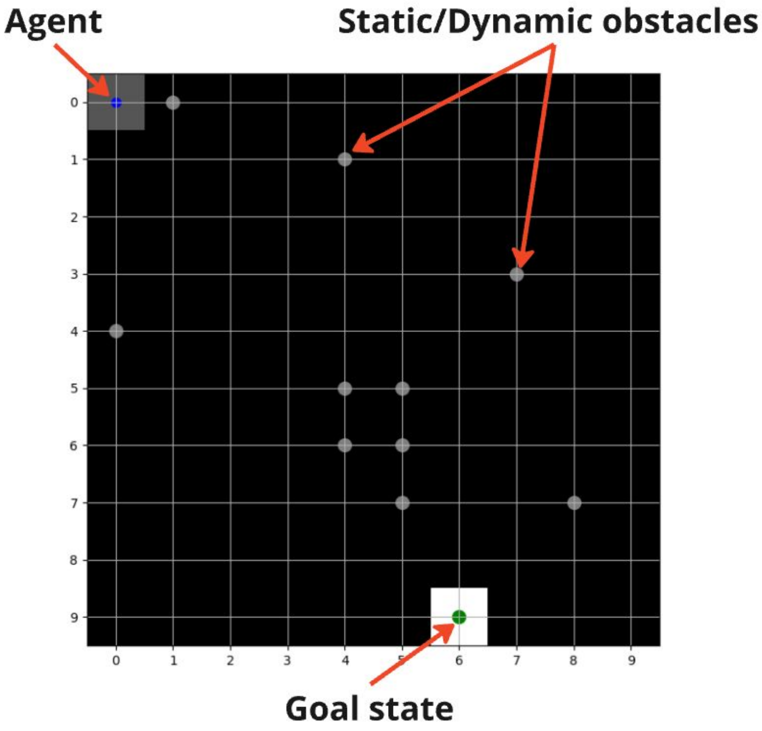
State: S = [x, y]

Actions: A = [up, down, left, right, right-up, right-down, left-up, left-down]

Environment: 2d grid with static / dynamic obstacles.

Rewards:

- Reached goal: +10
- Collision: -1
- For each move:
  - Horizontal: -0.1
  - Diagonal: -0.14



Steps of Q-Learning

Initialize the Q-table with zeros.

Repeat for each episode:

- Start at an initial state.
- Choose an action using an exploration-exploitation strategy (epsilon-greedy).
- Execute the action and observe the reward and the next state.
- Update the Q-value using the Q-learning update rule.
- Move to the new state.
- Repeat until the goal is reached or a maximum number of steps is taken.

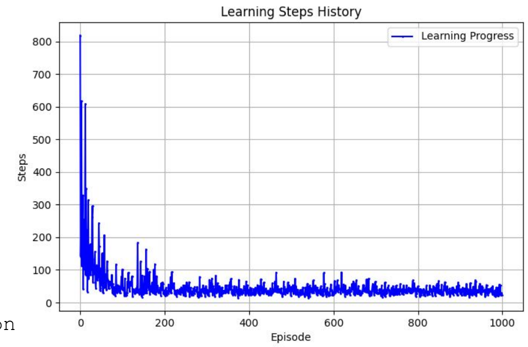
Use the learned Q-values to follow the optimal policy in future interactions.

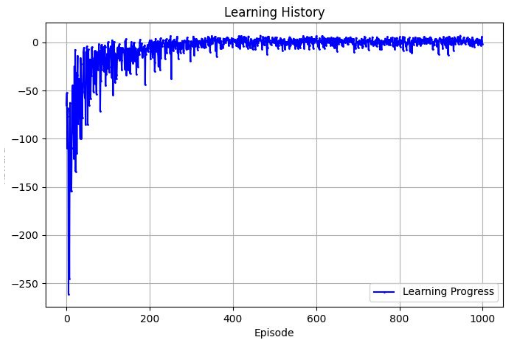
$$Q(s,a) \leftarrow \underset{\text{current values}}{Q(s,a)} + \overset{\text{learning rate}}{\alpha} \left(\overset{\text{reward}}{r} + \underset{\text{discount factor}}{\gamma} \overset{\text{Max Q}}{\max_{a'}} Q(\underset{\text{new state after taking action}}{s'}, a') - Q(s,a) \right)$$

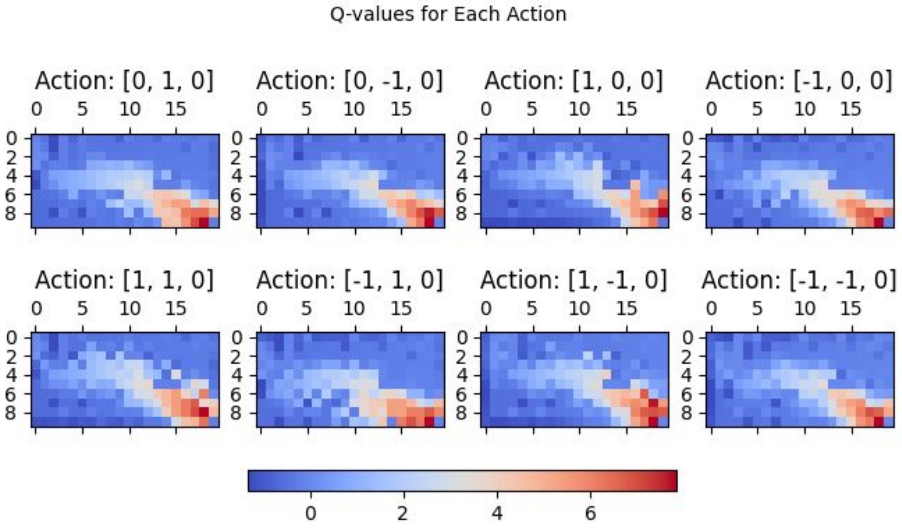
STATIC Obstacles

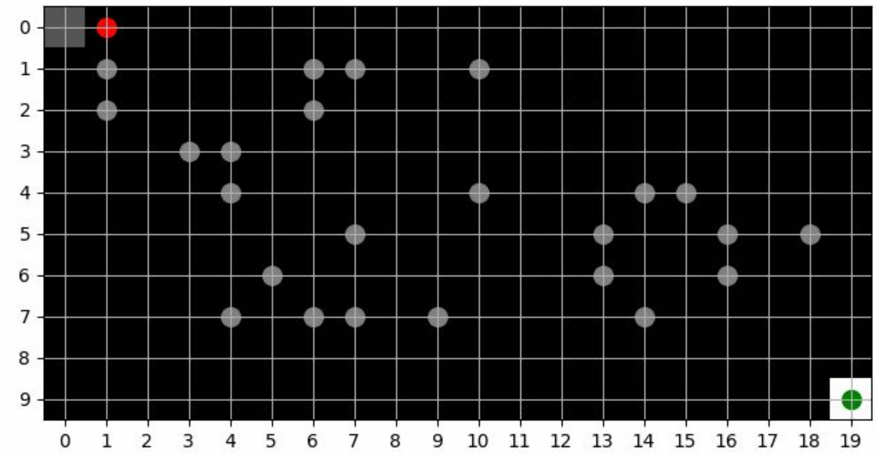
Environment: Static obstacles, no lidar
Actions: horizontal, vertical diagonal grid step

```
self.alpha = 0.1  # Learning rate
self.gamma = 0.9  # Discount factor
self.epsilon = 0.5  # Probability to take random action
self.iterations = 1000
```





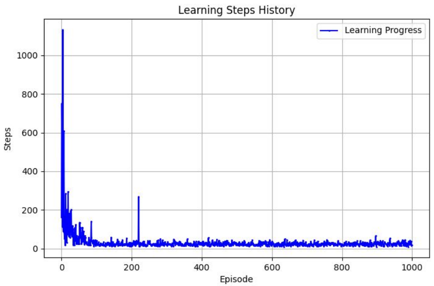


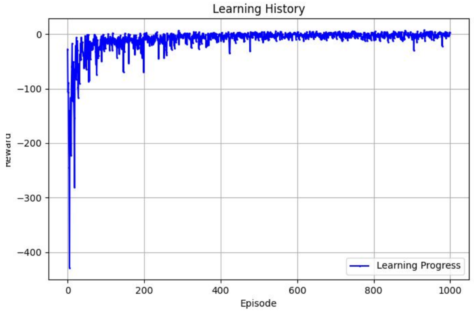


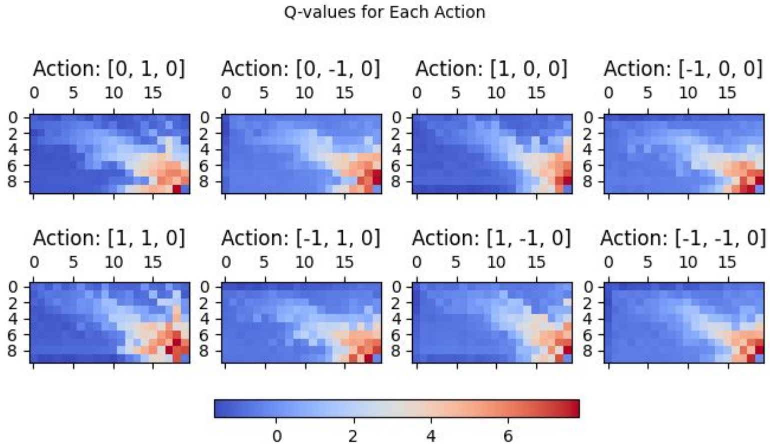
DYNAMIC Obstacles

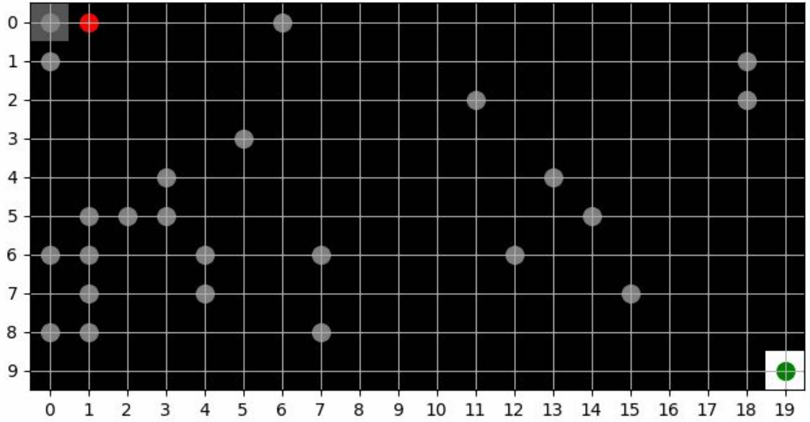
For a **grid environment** with **dynamic obstacles**, the Q-learning process handle:

- **State transitions** affected by moving obstacles.
- **Avoidance of collisions** by penalizing moves into occupied spaces.
- **Adapting to changes** in the environment over time.

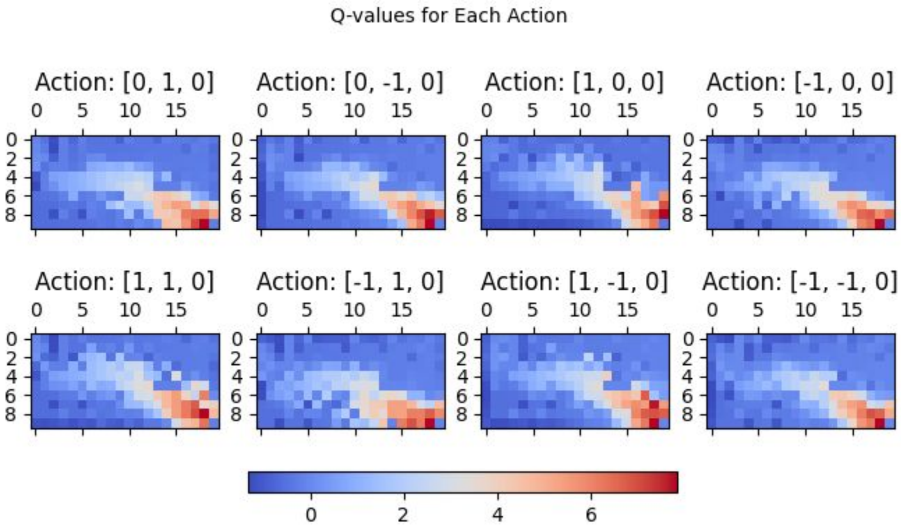




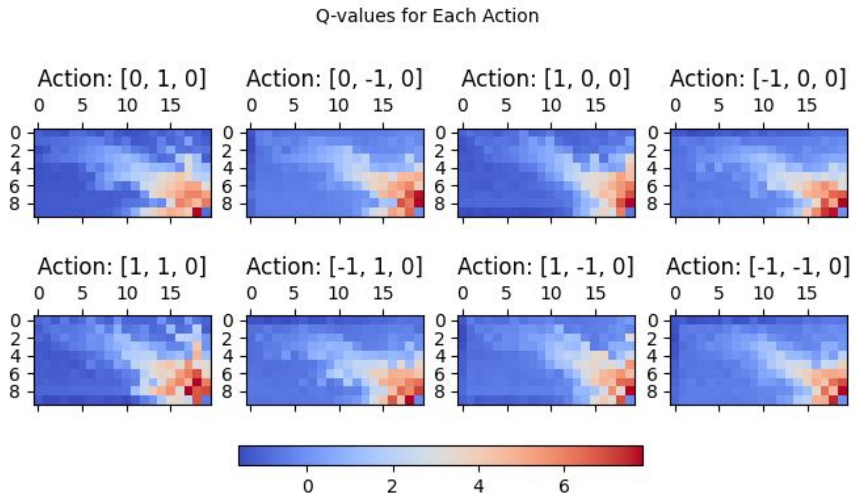




STATIC



DYNAMIC



Environment setup

The agent is equipped with a 2D Liar to detect and track dynamic obstacles within a 5x5 area.

State:

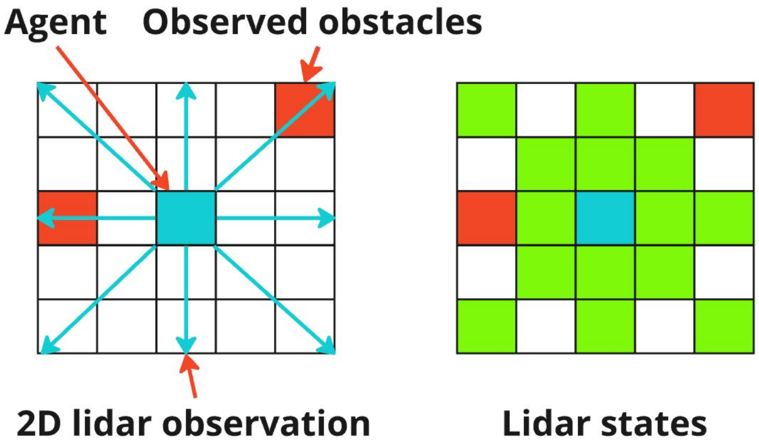
S = [observation(8 beems), dist to goal]

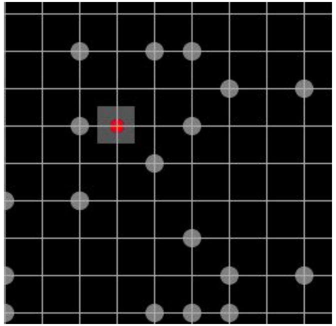
Actions: A = [up, down, left, right, right-up, right-down, left-up, left-down]

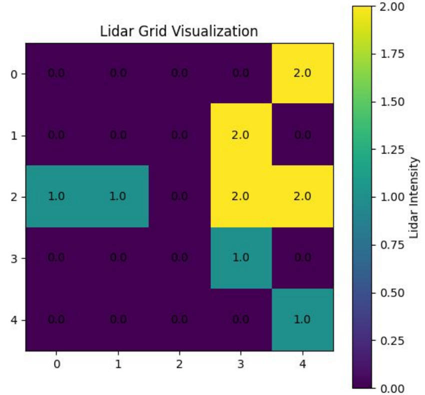
Environment: 2d grid with static / dynamic obstacles.

Rewards:

- Reached goal: +10
- Collision: -1
- If dist_to_goal become less: +0.1
- If dist_to_goal become more: -0.1







Self.lidar_state = [2, 2, 0, 0, 1, 0, 0, 1, 17.1]

Neural Network Architecture (DQN)

The architecture of the Deep Q-Network (DQN) consists of a **fully connected network** that maps input states to Q-values for different actions.

Network Structure:

- **Input layer**: Takes a state vector as input.
- **Hidden layer 1**: A fully connected layer with 128 units and a ReLU activation function.
- **Hidden layer 2**: Another fully connected layer with 128 units and a ReLU activation function.
- **Output layer**: A fully connected layer that outputs Q-values corresponding to each possible action.

```
# Learning parameters:
GAMMA = 0.99
LR = 0.001
EPSILON = 1.0 #at beginning
EPSILON_DECAY = 0.995
EPSILON_MIN = 0.01
BATCH_SIZE = 64
MEMORY_SIZE = 10000
TARGET_UPDATE = 10
EPOCH = 650
```

Q-Values Calculation

Q-values represent the expected future rewards for each action at a given state. The **Q-value** for a state-action pair is updated using the **Bellman equation**:

reward

$$Q(s_t, a_t) = R_t + \gamma \cdot \max_{a'} Q(s_{t+1}, a')$$

current Q-value for state S and action a

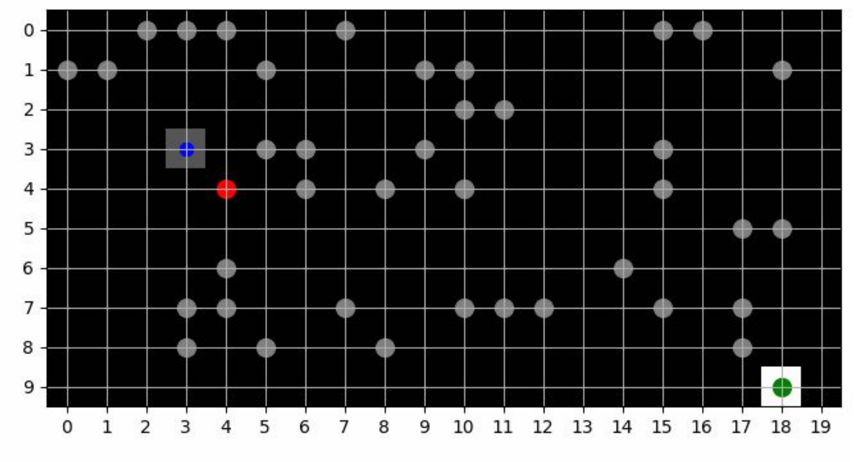
discount factor

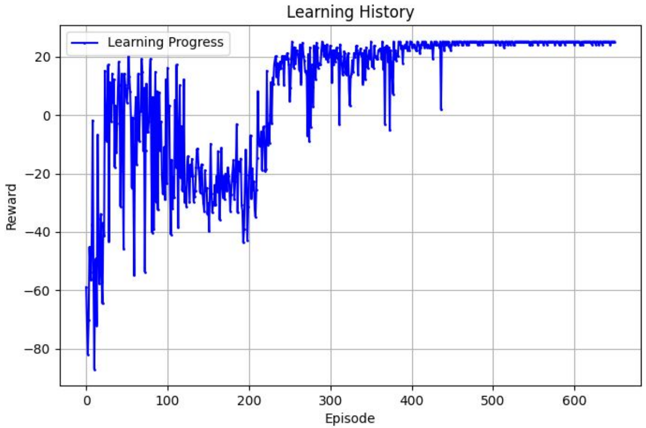
max Q-value for next state across all possible action

LIDAR. DQN STATIC

Environment: Static obstacles, Lidar active

Actions: horizontal, vertical diagonal grid step



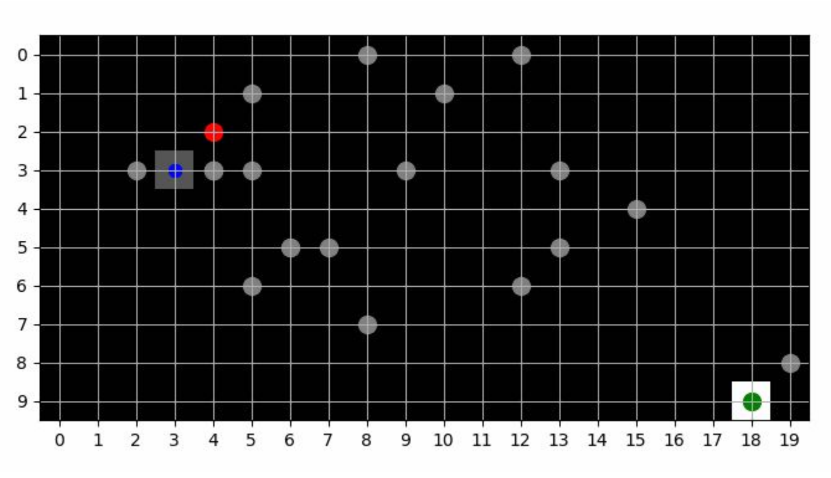


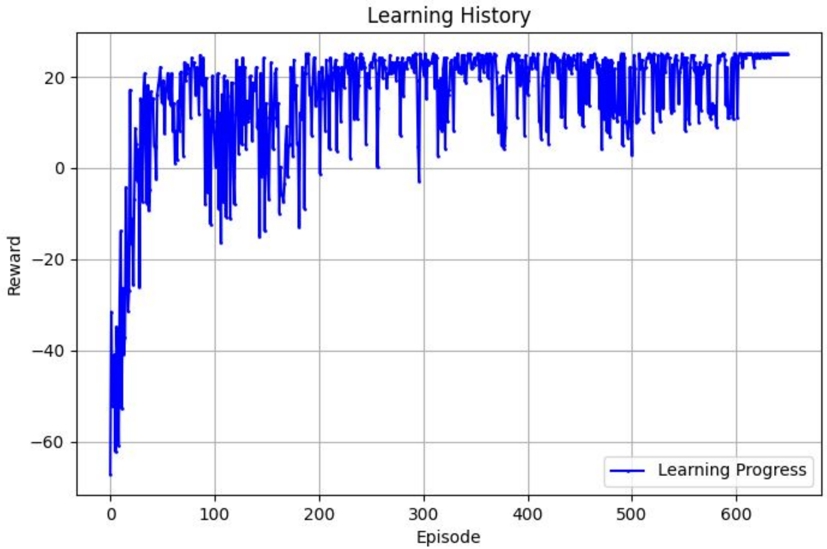
agent did: 16 steps. Reward: 25.15549442140351, last action:[0, 1]

LIDAR. DQN DYNAMIC

Environment: Dynamic obstacles, Lidar active

Actions: horizontal, vertical diagonal grid step





agent did: 16 steps. Reward: 25.15549442140351