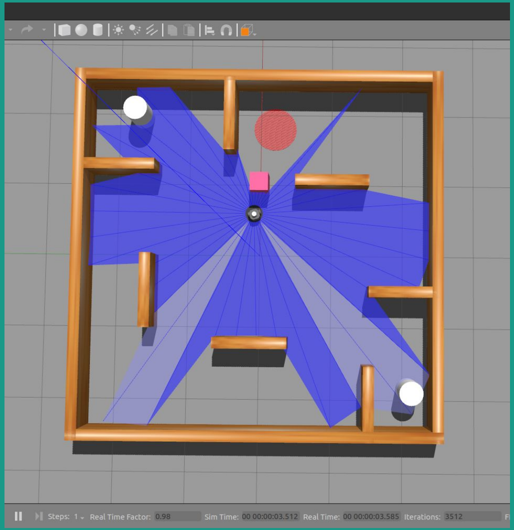
08/10 Week 10 Final Project Report

Enhancing Robotic Navigation: An Evaluation of Single and Multi-Objective Reinforcement Learning Strategies

Vicki Young, Jumman Hossain

Problem Statement

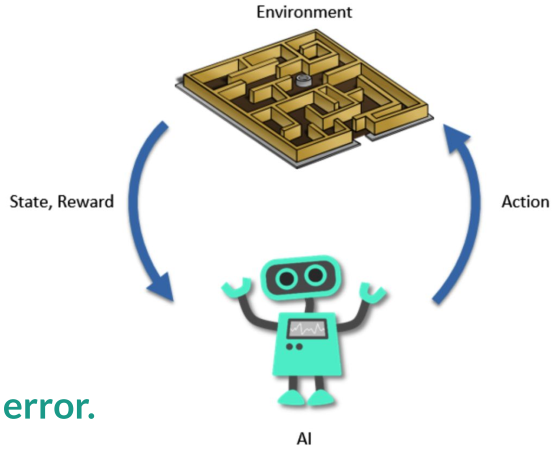
Goal-driven autonomous navigation:
training robots how to get to a specific place (end goal) while smartly avoiding obstacles



A comparative analysis between **single-objective and multi-objective reinforcement learning (RL) methods** for training a robot in goal-driven autonomous navigation

- **In traditional RL:** the robot learns to optimize a single objective.
    - This **fails when there are multiple, conflicting objectives** that need to be considered.
- Solution: **multi-objective RL.**

Background Information



Reinforcement Learning (RL)

RL is best for **autonomous robot performance**.

- Similar to how humans learn: **from experience by trial and error.**

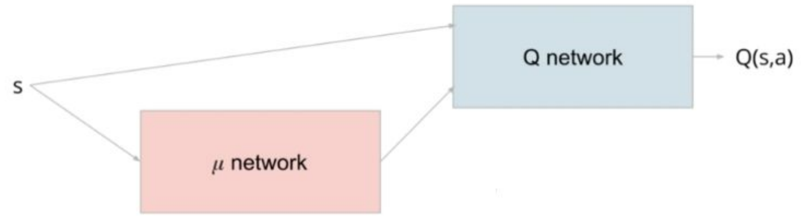
An agent learns by taking actions to interact with its environment and receives feedback in the form of a reward.

Agent’s goal: **maximize the total reward** it receives.

Learn to choose the best actions in given situations which lead to the overall best outcome.

- **Policy** = Taking a particular action in a particular situation.
- **Optimal policy** = Taking the best/optimal action.

The goal of RL is essentially to learn the optimal policy.



Single-Objective Reinforcement Learning

Traditional RL methods: an agent learns to prioritize a **single objective** by aiming to maximize a single numerical reward. **This study:** the robot has one objective, navigate to the end goal.

- Deep Q-Network (DQN) = learns a **Q-value function,** which informs the action quality of a given action (its Q-value).
- Deep Deterministic Policy Gradient (DDPG) = learns a policy, a way of deciding what actions to take in a given situation. Uses an **actor-critic algorithm**, where an actor chooses actions and a critic gives their Q-values.
- Twin Delayed DDPG (TD3) = based on DDPG, has an actor and two critics, and also learns a policy. Helps **avoid overestimating the value of a bad action**, since the lower Q-value will be used instead.

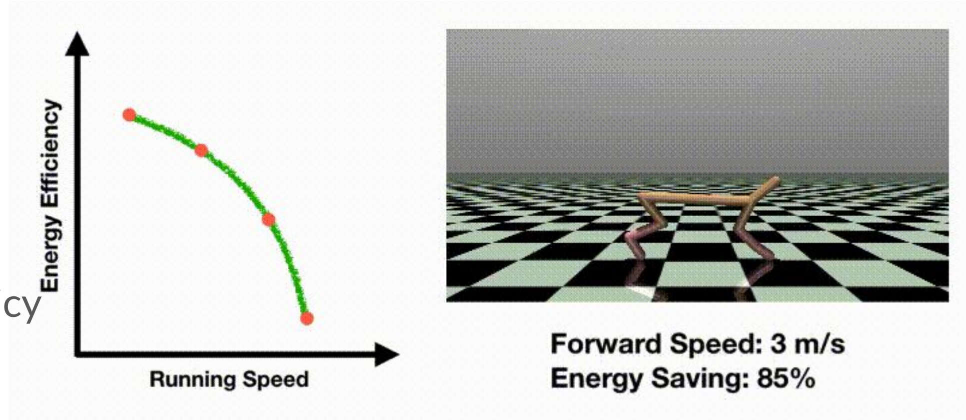
Multi-Objective Reinforcement Learning (MORL)

Real-world problems often require balancing multiple, potentially conflicting objectives.

Example: A self-driving car. The car must reach its destination as fast as possible, but must also avoid collisions and minimize energy consumption.

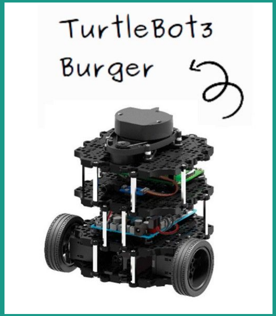
MORL simultaneously optimizes multiple objectives.

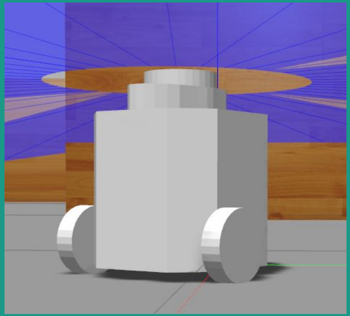
- **Modify the reward function:** return a vector of rewards
- **Find a Pareto optimal solution:** no policy is strictly the best



Methods

Frameworks

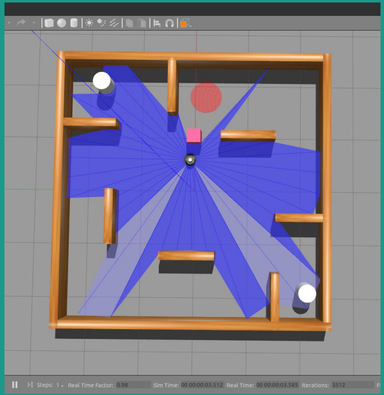


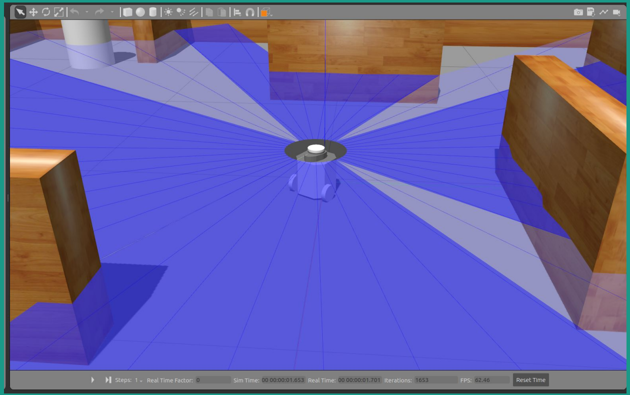


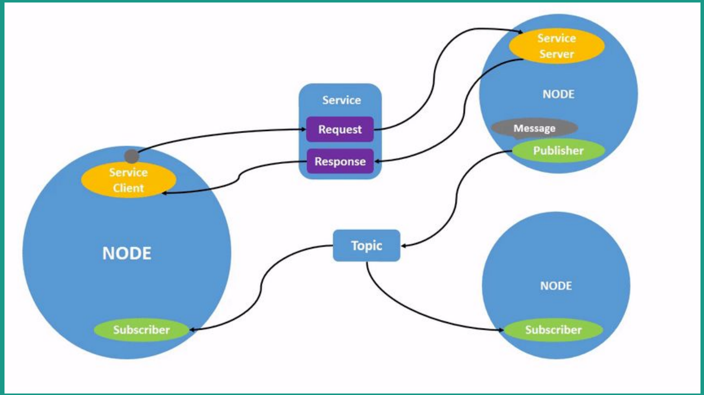
Gazebo simulation framework is used to simulate testing/training the robot.

TurtleBot3 robot reads data from **LiDAR sensors** using **ROS topics.**

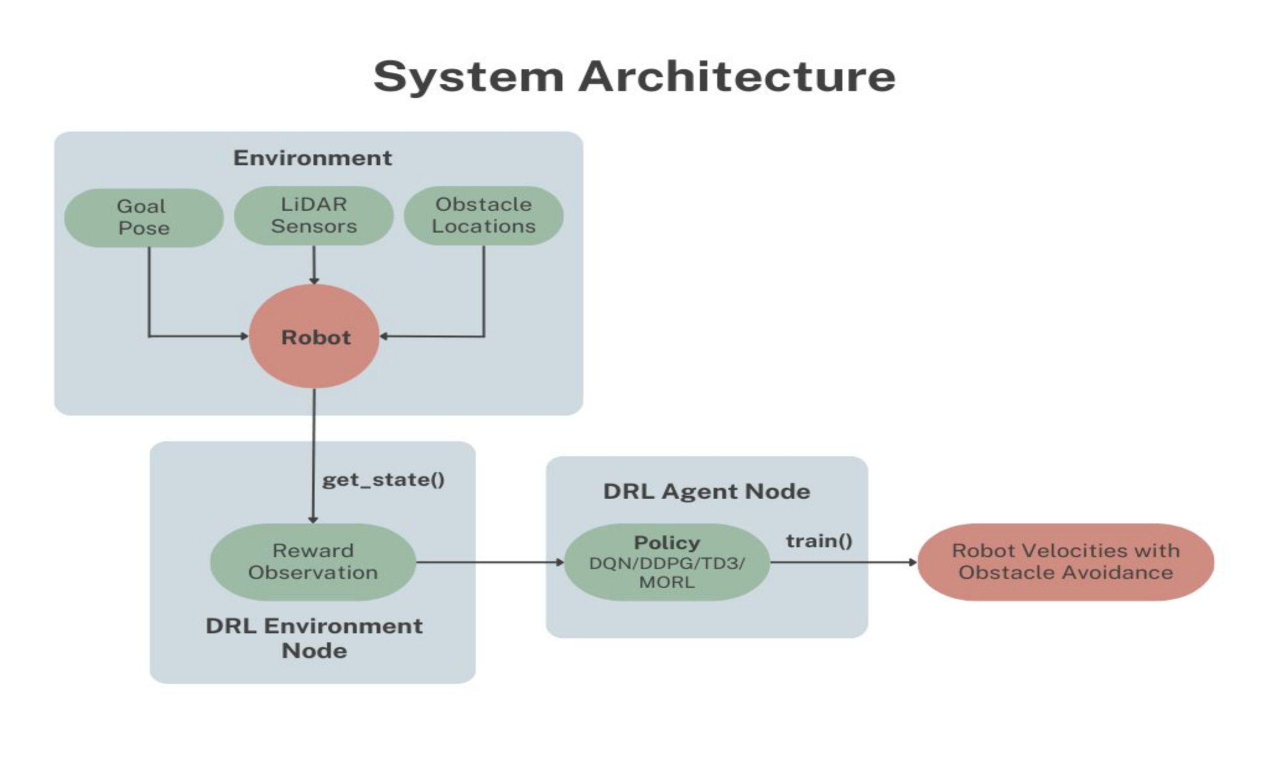
Robot Operating System (ROS) control features allow RL methods to use the robot sensory information to make **navigation decisions.**







System Architecture



The system architecture describes how **information about the environment is input to the reward function**, and **its output** helps the chosen policy **decide what action the robot should take**.

Reward Function

robot movement = `action_linear` & `action_angular`

robot distance from end goal = `goal_dist` & `goal_angle`

robot distance from nearest obstacle = `min_obstacle_dist`

robot status in terms of objective = `succeed`

1. Yaw Reward: $r_{yaw} = -1 \cdot |\text{goal\_angle}|$

2. Angular Velocity: $r_{vangular} = -1 \cdot (\text{action\_angular}^2)$

3. Distance Reward: $r_{distance} = \frac{1}{2} \cdot \left(\frac{2 \cdot \text{goal\_dist\_initial}}{\text{goal\_dist\_initial} + \text{goal\_dist}}\right) - 1$

4. Obstacle Reward: $r_{obstacle} = -1 \cdot \begin{cases} -20 & if \ \text{min\_obstacle\_dist} < 0.22 \\ 0 & otherwise \end{cases}$

5. Linear Velocity Reward: $r_{vlinear} = -2 \cdot ((0.22 - \text{action\_linear}) \times 10)^2$

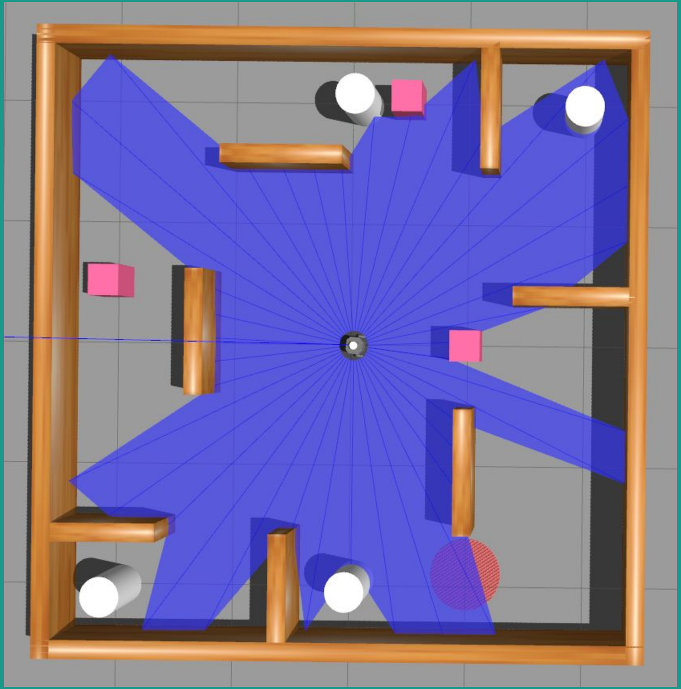
6. Total Reward without Success Condition:

$$\text{reward} = r_{yaw} + r_{vangular} + r_{distance} + r_{obstacle} + r_{vlinear} - 1$$

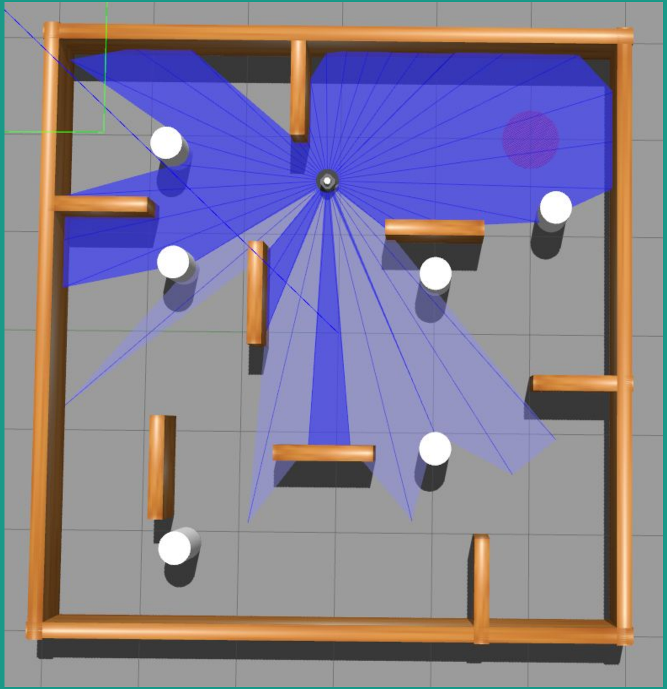
7. Final Reward with Success Condition:

$$\text{reward} = \begin{cases} \text{reward} + 2500 & if \ \text{succeed} = \text{SUCCESS} \\ \text{reward} - 2000 & if \ \text{succeed} = \text{COLLISION\_OBSTACLE or COLLISION\_WALL} \\ \text{reward} & otherwise \end{cases}$$

Environments



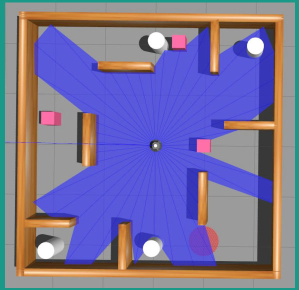
Stage A has an environment with **3 non-moving obstacles** (pink) and **4 moving obstacles** (white).



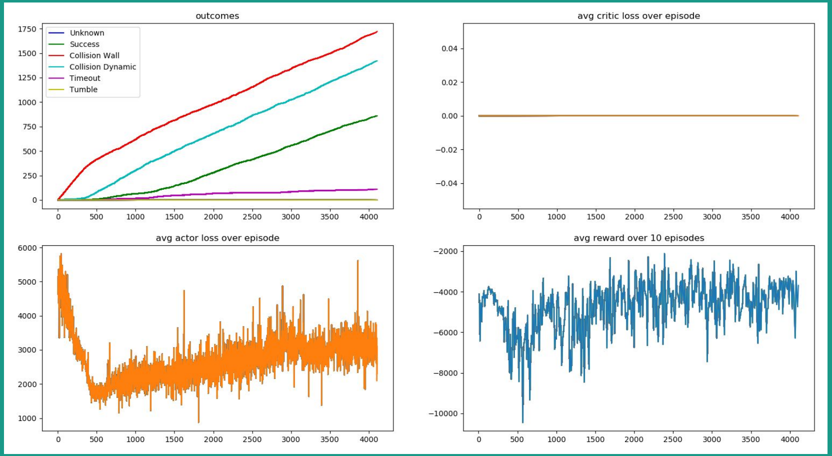
Stage B has an environment with **6 moving obstacles** (white).

Results

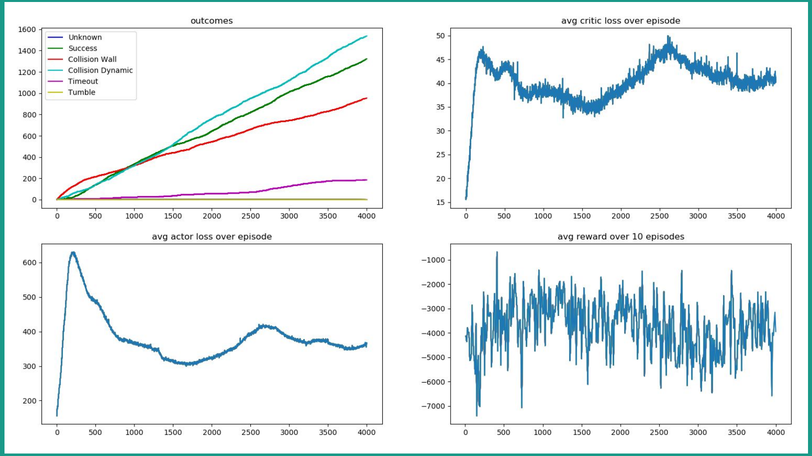
Training algorithms on stage A



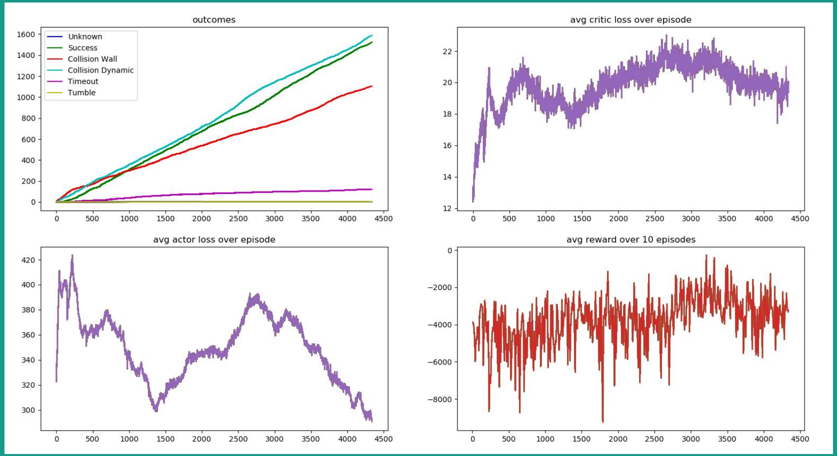
DQN



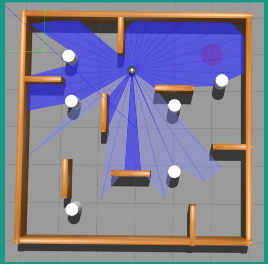
TD3



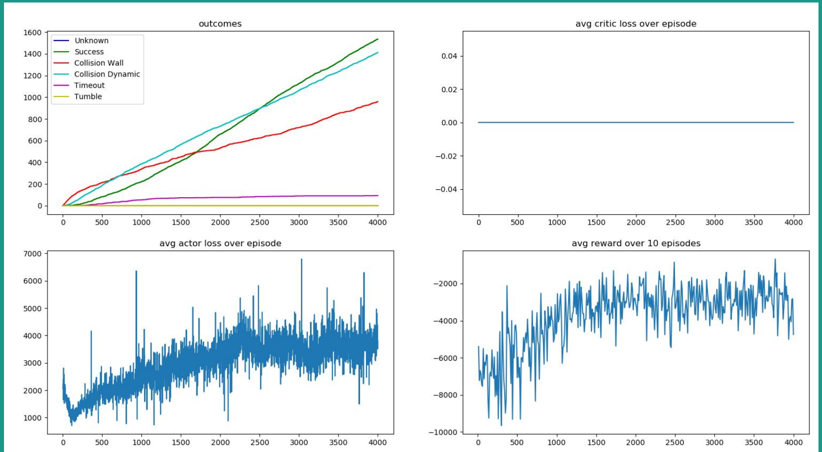
DDPG



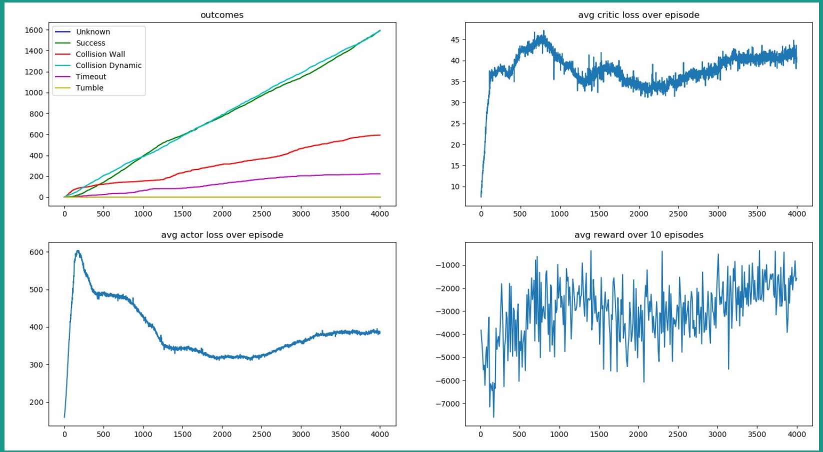
Training algorithms on stage B



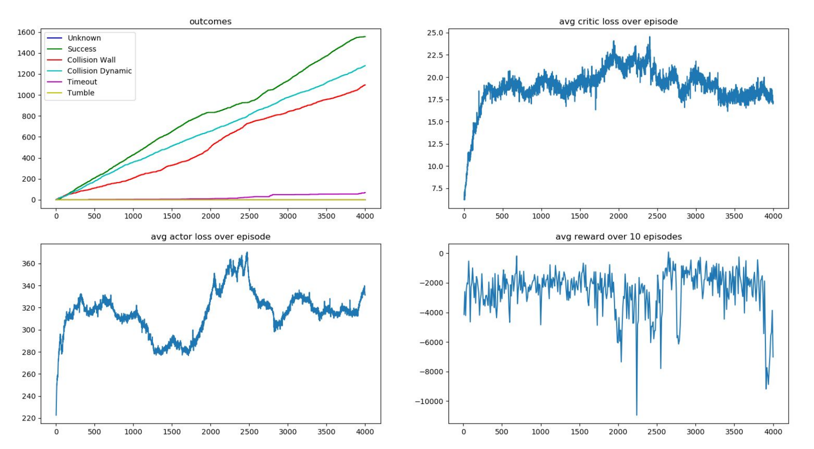
DQN



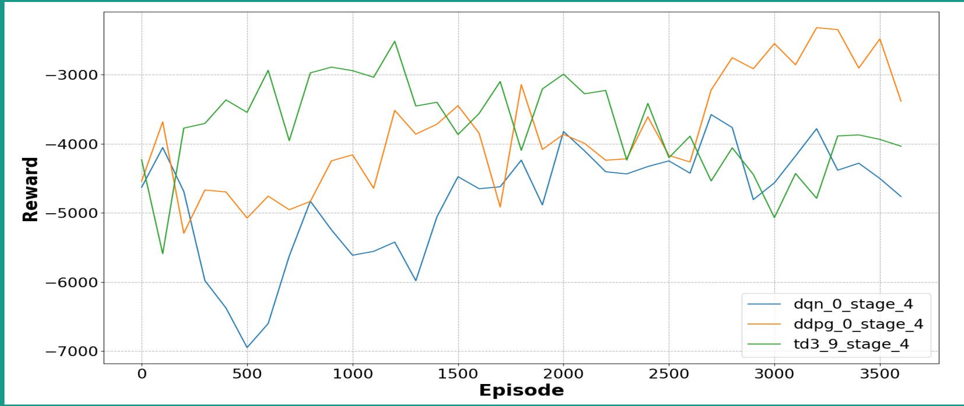
TD3

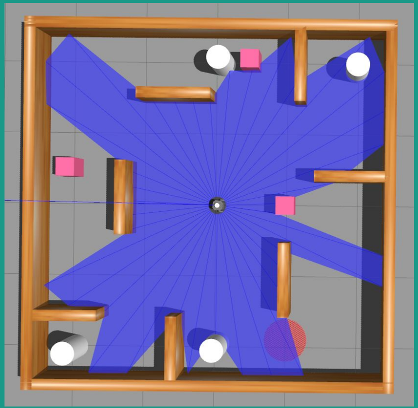


DDPG



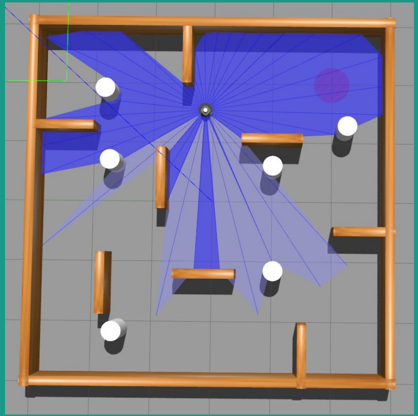
Reward graph comparisons

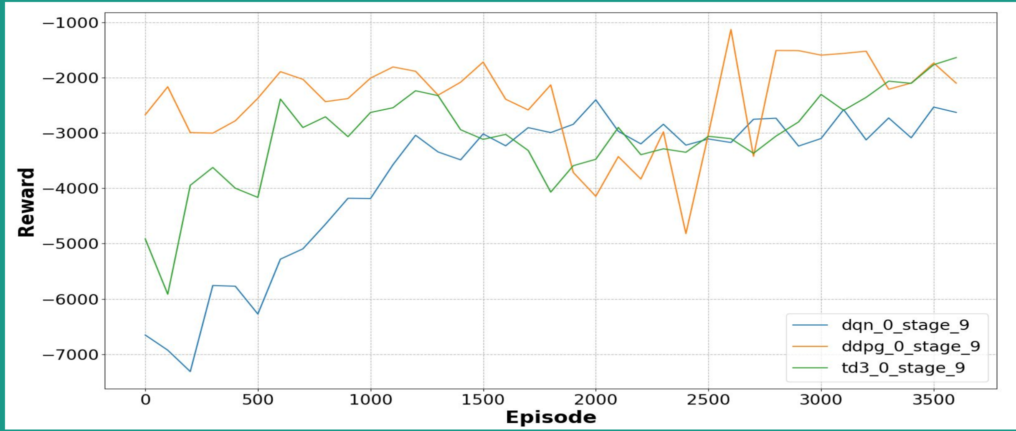




stage A

stage B



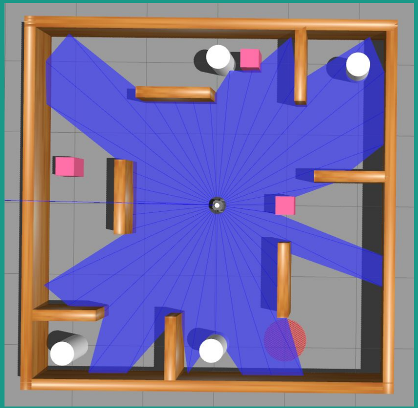


Reward graph comparisons

DQN best performing episodes: [2700, 2800, 3200, 2000]
with scores: **[-3579, -3766, -3782, -3823]**

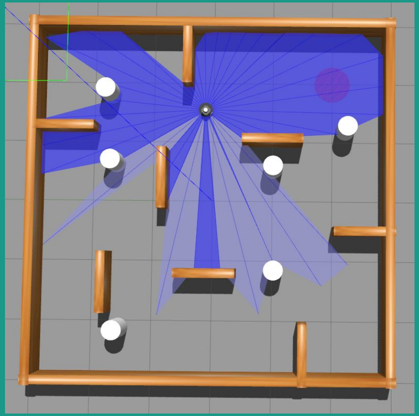
DDPG best performing episodes: [3200, 3300, 3500, 3000]
with scores: **[-2322, -2349, -2484, -2551]**

TD3 best performing episodes: [1200, 900, 600, 1000]
with scores: **[-2517, -2892, -2938, -2943]**



stage A

stage B



DQN best performing episodes: [2000, 3500, 3100, 3600]
with scores: **[-2401, -2532, -2578, -2629]**

DDPG best performing episodes: [2600, 2800, 2900, 3200]
with scores: **[-1126, -1509, -1512, -1522]**

TD3 best performing episodes: [3600, 3500, 3300, 3400]
with scores: **[-1637, -1766, 2064, -2103]**

Testing algorithms

DDPG_stage_A testing on stage_A (episode 4200 to 8000 | 3800 total)
Successes: 1450 (38.16%), collision (wall): 543 (14.29%), collision (obs): 1768 (46.53%), timeouts: 39, (1.03%), tumbles: 0, (0.00%

DDPG_stage_A testing on stage_B (episode 4000 to 4200| 200 total)
Successes: 31 (15.50%), collision (wall): 100 (50.00%), collision (obs): 51 (25.50%), timeouts: 18, (9.00%), tumbles: 0, (0.00%)

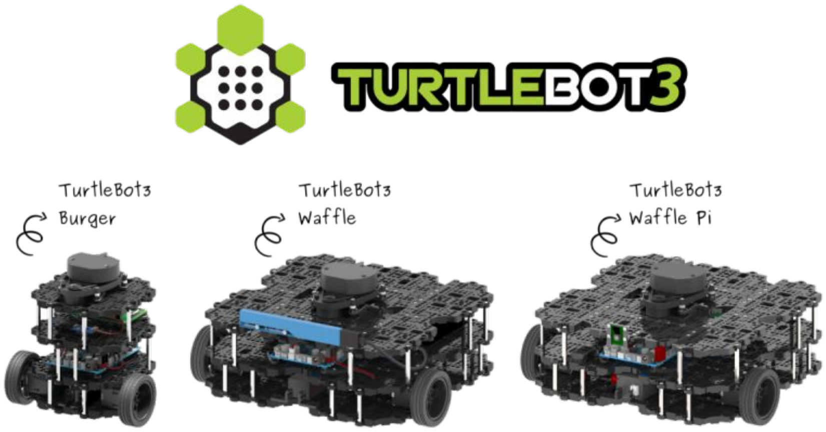
TD3_stage_A testing on stage_A (episode 4000 to 5700 | 1700 total)
Successes: 584 (34.35%), collision (wall): 301 (17.71%), collision (obs): 593 (34.88%), timeouts: 222, (13.06%), tumbles: 0, (0.00%)

TD3_stage_A testing on stage_B (episode 4000 to 4100 | 100 total)
Successes: 42 (42.00%), collision (wall): 16 (16.00%), collision (obs): 34 (34.00%), timeouts: 8, (8.00%), tumbles: 0, (0.00%)

Conclusion

Future work

- Continue to test the trained models
  - Work on increasing the success rate
- Implement MORL algorithm
  - Compare the performance of MORL with single-objective RL algorithms
- Load models onto a physical robot
  - Test on different robots?



Skills acquired during the REU

- Experience with machine learning—**reinforcement learning**
  - RL algorithms: MDP, V & Q-value, MORL
  - Deep neural networks
- Experience with **Gazebo**, **ROS**, Python libraries
  - ROS packages, topics, services/clients
  - **numpy, pytorch, rclpy**
- Experience with **Overleaf** and **LaTeX**

$$
\begin{aligned}
q_\pi(s,a) &\doteq \mathbb{E}_\pi\left[G_t \mid S_t = s, A_t = a\right] \\
&= \sum_{s'} \sum_{r} p(s',r \mid s,a)\Big[r + \gamma \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s'\right]\Big] \\
&= \sum_{s'} \sum_{r} p(s',r \mid s,a)\left[r + \gamma \sum_{a'} \pi(a' \mid s') \mathbb{E}_\pi\left[G_{t+1} \mid S_{t+1} = s', A_{t+1} = a'\right]\right] \\
&= \sum_{s'} \sum_{r} p(s',r \mid s,a)\left[r + \gamma \sum_{a'} \pi(a' \mid s') q_\pi(s',a')\right]
\end{aligned}
$$

Research experience gained during the REU

Data collection is 80% of the work

- Reading related papers
    - Draw connections
- Training/testing process
- Research equipment needed
- How to develop + present ideas
    - Charts, graphs, equations

```python
def get_reward_A(succeed, action_linear, action_angular, goal_dist, goal_angle, min_obstacle_dist):
        # [-3.14, 0]
        r_yaw = -1 * abs(goal_angle)

        # [-4, 0]
        r_vangular = -1 * (action_angular**2)

        # [-1, 1]
        r_distance = (2 * goal_dist_initial) / (goal_dist_initial + goal_dist) - 1

        # [-20, 0]
        if min_obstacle_dist < 0.22:
            r_obstacle = -20
        else:
            r_obstacle = 0

        # [-2 * (2.2^2), 0]
        r_vlinear = -1 * (((0.22 - action_linear) * 10) ** 2)

        reward = r_yaw + r_distance + r_obstacle + r_vlinear + r_vangular - 1

        if succeed == SUCCESS:
            reward += 2500
        elif succeed == COLLISION_OBSTACLE or succeed == COLLISION_WALL:
            reward -= 2000
        return float(reward)
```

Thank You