Cooperative Multi-Robot Navigation in Dynamic Environment with Deep Reinforcement Learning

Seminar on Artificial Intelligence II

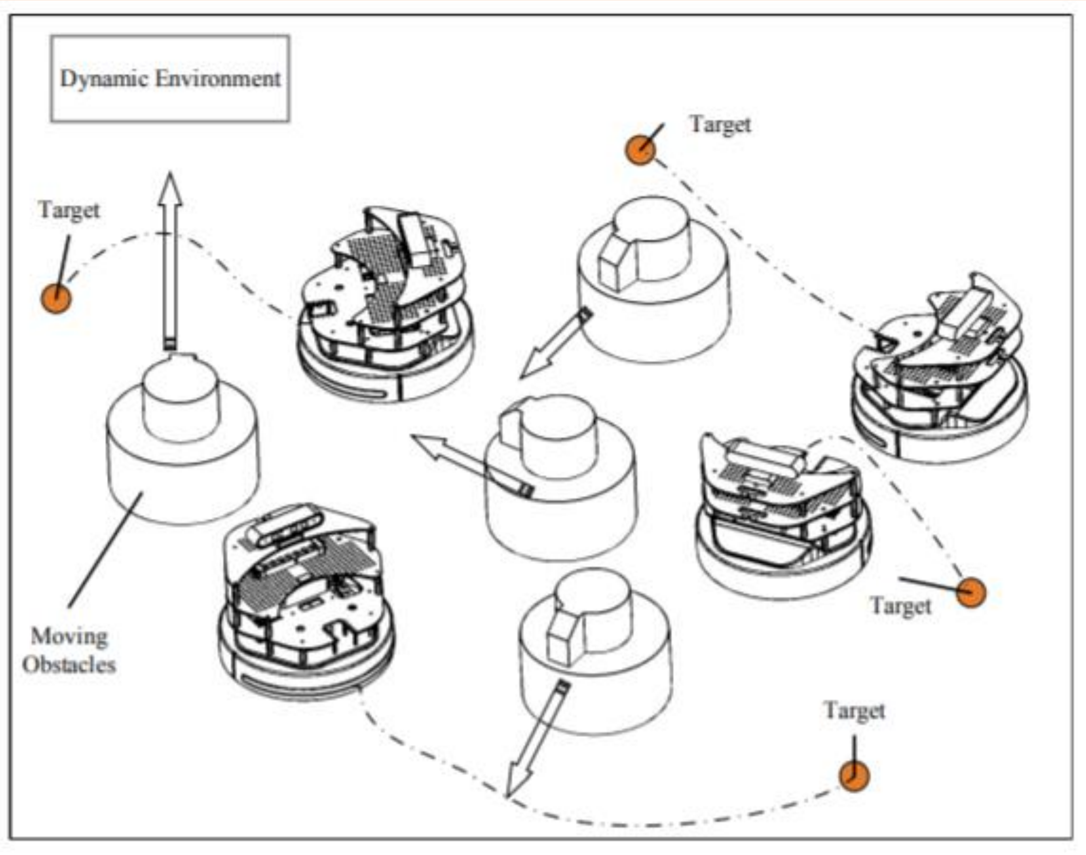
Gabriela Suchopárová, 16. 3. 2022

Outline

1. Introduction
2. Cooperation framework design
3. Deep reinforcement learning model
4. Target location allocation algorithm
5. How to solve the transfer to real world
6. Experiments

Multi-robot navigation problem

- N robots, M obstacles
  - Obstacles **move** as well
- Dynamic
- Partially observable
- **Multiple targets**
  - Robots can go to any target
  - Goal allocation
- Simulation and **Real World**
- Task: learn the **navigation policy**

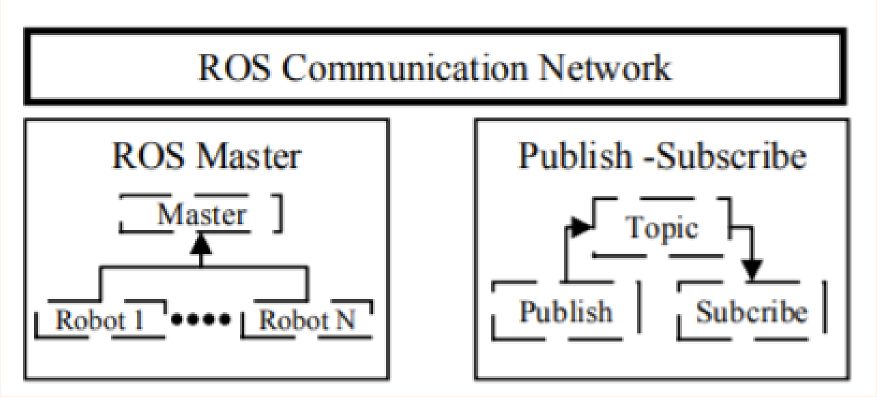


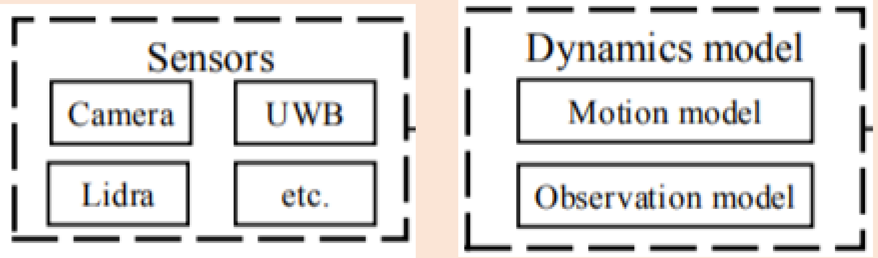
Challenges

- Efficient **target** location **allocation**
  - Allocate the goals fast
  - Reduce the total travel time
- Robot **cooperation**
  - How to combine the experience of all robots
- **Transfer** from **simulation** to **real world**
  - Noise in observations from sensors
  - Noise in motion when applying actions
  - The noise parameters differ across different scenarios (sensor type etc.)

System framework of multi-robot navigation

- They **share one neural network** and policy
  - The network is trained using input from all robots
- Agents communicate through ROS
  - They **share observations**
    - obstacle positions, their position,…
  - They **receive actions**
- Agents get information from **sensors**
- Agents have a **dynamics model**





Problem definition

- POMDP
- States
  - **Robot** states: $\boldsymbol{s}_r^t = \left[p_{rx}^t, p_{ry}^t, \theta_r^t, v_{rx}^t, v_{ry}^t\right]$ (position, orientation, velocity)
  - **Obstacle** states: $\boldsymbol{s}_o^t = \left[p_{ox}^t, p_{oy}^t, v_{ox}^t, v_{oy}^t, r_o^t\right]$ (position, velocity, radius)
  - **Target** positions: $\boldsymbol{s}_g = \left[p_{gx}, p_{gy}\right]$
- Actions: $\boldsymbol{a}^t = [v_t^t, v_r^t] \sim \pi\left(\boldsymbol{a}^t \middle| \boldsymbol{s}_r^t, \boldsymbol{s}_o^t, \boldsymbol{s}_g\right)$
- Objective:
  - **Minimize travel time** of all agents **to goals** while **avoiding collisions**

$$
\begin{aligned}
\underset{\pi_\theta}{\operatorname{argmin}} \quad & \mathbb{E}\left[T \middle| \pi_\theta, \mathbf{s}_{r,1:N}, \mathbf{s}_{o,1:M}, \mathbf{s}_{g,1:N}\right] \\
s.t. \quad & \forall i, j \in [1, N], k \in [1, M] \\
& d_{rr,i,j} > 2r_r \\
& d_{ro,i,k} > r_r + r_o \\
& d_{g,i} < d_{\min}
\end{aligned}
$$

RL quick overview

- Estimate v(s) or q(s,a) … average return from state s (starting with a)
  - $V(s) = \mathbb{E}[G_t \mid s_t = s] = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 r_{t+3} + \ldots \mid s_t = s]$
- Bellman equation
  - $V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P(s', r \mid s, a)(r + \gamma V^\pi(s'))$
- The equation is estimated, since we don't know the probability model
  - In POMDP, we even don't know the states
  - We also want to learn the policy π

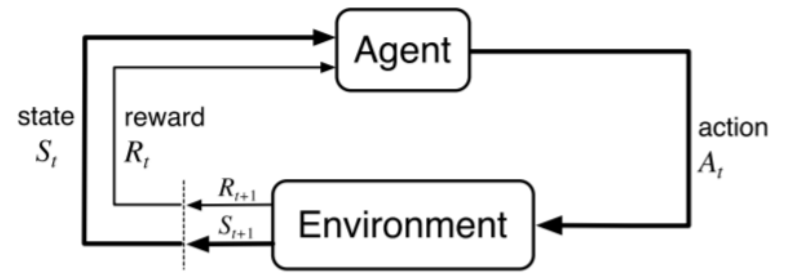


Figure 3.1 of "Reinforcement Learning: An Introduction, Second Edition".

1. Deep reinforcement learning framework

- **State** space: $\boldsymbol{s}^t = [\boldsymbol{s}_r^t, \tilde{\boldsymbol{s}}_r^t, \boldsymbol{s}_o^t, \boldsymbol{s}_g^t]$
  - Robot state, other robot states, obstacle state, *allocated* goal state
- **Action** space: $\boldsymbol{a}^t = [v_t^t, v_r^t]$
  - Velocities **clipped** to some range (hyperparameter)
  - The reason is limited obstacle detection speed
- **Reward** of robot *i*: $r_i^t = r_{g,i}^t + r_{c,i}^t$

$$r_{g,i}^t = \begin{cases} 5 & if d_g^t < 0.1 \\ 1 + d_g^t * 0.5 & if 0.1 < d_g^t < 0.4 \\ 10 * (d_g^{t-1} - d_g^t) & otherwise \end{cases}$$

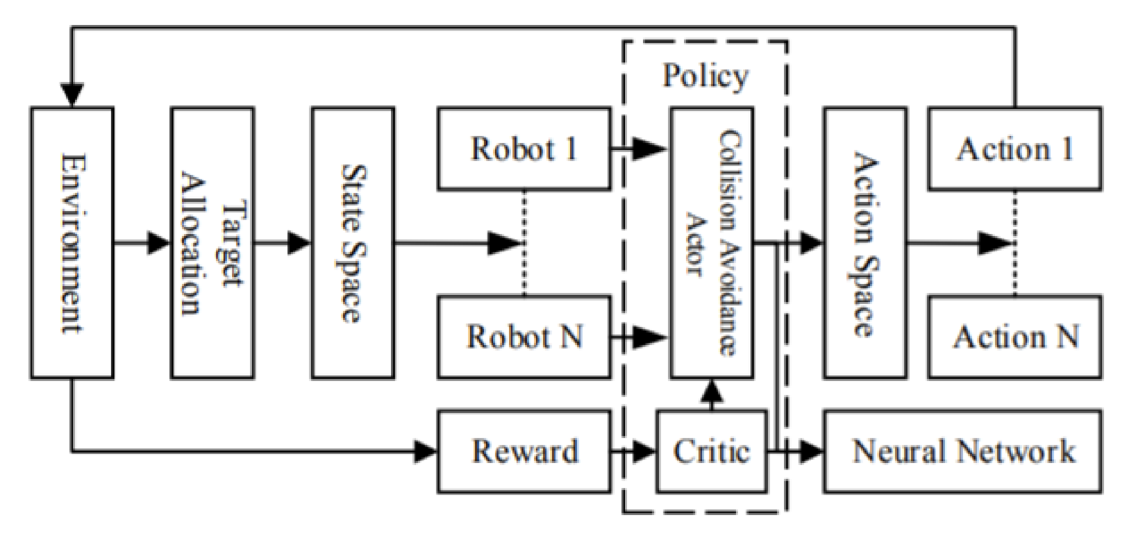
Goal reward

$$r_{c,i}^t = \begin{cases} -10 & if \quad d_{rr}^t < 2 * r_r \\ -10 & if \quad d_{ro}^t < r_r + r_o \\ 0 & otherwise \end{cases}$$

Collision avoidance reward

1. Deep reinforcement learning framework

- Model used – **PPO**
- Temporal-difference method
- Actor-critic – 2 dense networks
  - Actor – outputs **actions**
  - Critic – **estimates** the **value** function



- Predicted actions are **continuous**
  - Transitional and rotational velocities are modelled
  - Network outputs mean and standard deviation (Gaussian distribution)
  - The **action** is **sampled** from these distributions

2. Target location allocation

- *Why?* – While avoiding obstacles, the **closest target** may **change**
- For N robots and N targets, allocate targets such that the total **distance** is **minimal**
  - Allocation is a permutation of robots
- Presented algorithm – basically just **check all permutations** and select the best one
- Run only **after T_a steps**
  - T_a is a hyperparameter
  - Chosen according to no. of agents & computing power

3. Dynamics randomization

- Transfer from simulation to the real world
- Real sensors have noise that can be different for every device
- We want to **avoid retraining** the model on new data
- However, not retraining could lead to worse performance
- Solution: **add noise** during **training**
  - The noise stays the same during the episode
  - Uniformly selected from a predefined range
  - All noises are $N(0, \xi_i)$
  - At each step, we sample from it

- Noise in transitional velocity, ξ_1
- Noise in rotational velocity, ξ_2
- Noise in the position (coordinates) of robots, ξ_3
- Noise in position (coordinates) of obstacles, ξ_4
- Noise in the measurements of obstacles ξ_5
- Mass of robots m_r

Full training cycle

- Classic DRL setting
  - Simulate multiple episodes
- For every episode, **set dynamics**
- Train
  - Every few steps, **allocate goals**
  - Sample actions using the **actor**
  - Collect **next state**, **reward**
- After T steps, compute **losses**

Algorithm 2 Policy Training with PPO

```
Initialize neural network π_θ
for episode=1,2,... do
    Reset the environment with the initial state, s_init
    Sample the dynamics parameters λ from a range γ uniformly, λ ~ γ
    for robot i=1,2,... do
        Receive state s_i^t, select the goal position s_{g,i}^t
        Add noise, ŝ_i^t ~ s_i^t + [ξ_3, ξ_4, ξ_5]
        Sample action a_i^t ~ π_θ(a_i^t | ŝ_i^t)
        Add noise, â_i^t ~ a_i^t + [ξ_1, ξ_2]
        Publish â_i^t to robot i
        Collect state ŝ_i^t, reward r_i^t and a_i^t for T_i time steps
        Compute advantage estimates Â_i^1, ..., Â_i^T
    end for
    Optimize surrogate loss L^CLIP(θ) wrt θ, with Adam optimizer and learning rate l_a for K epochs
    θ_old ← θ
    Optimize value loss L^V(ψ) wrt ψ, with Adam optimizer and learning rate l_v for L epochs
    ψ_old ← ψ
end for
```

Related work

1. DRL models
    - Q-learning, DQN, A3C, DDPG, PPO
    - The difference is what they estimate (state/action-value function) and how
2. Multi-agent learning
    - MADDPG – shared critic, one actor per robot
    - SLCAP – similar approach, but PPO
    - GA3C-CADRL – one policy for all agents, asynchronous
    - These methods need preallocated targets
    - IDRL – allocates targets, but needs static obstacles

High hardware cost,
needs specific sensors

Asynchronous update
inefficient for
homogenous robots

3. Real world transfer – either retrain, or complicated dynamics model

Experiments

- Simulation
  - Gym-Gazebo env, Turtlebot robots
  - Obstacles – turtlebots that periodically change velocity
- Three models
  - 1. Without target allocation and dynamics
  - 2. With target allocation
  - 3. Full model
- Real world scenario
  - UWB, Kinect v2 – coordinates, obstacle pos.

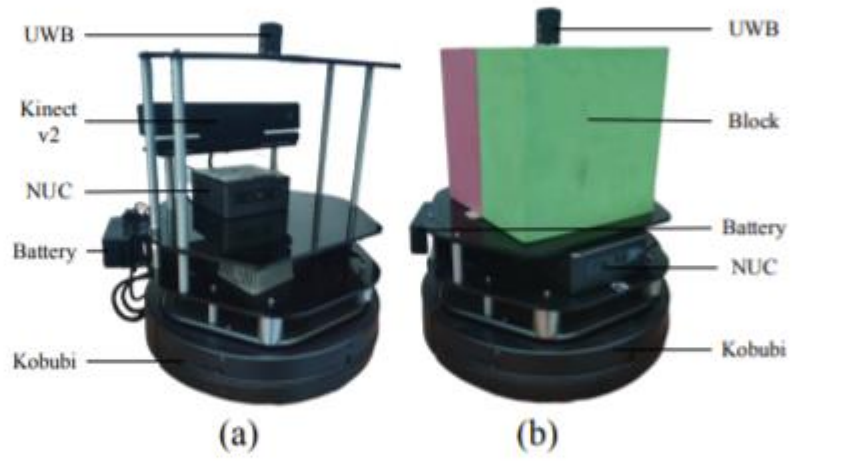


Fig. 5. Configuration of real robots and obstacles

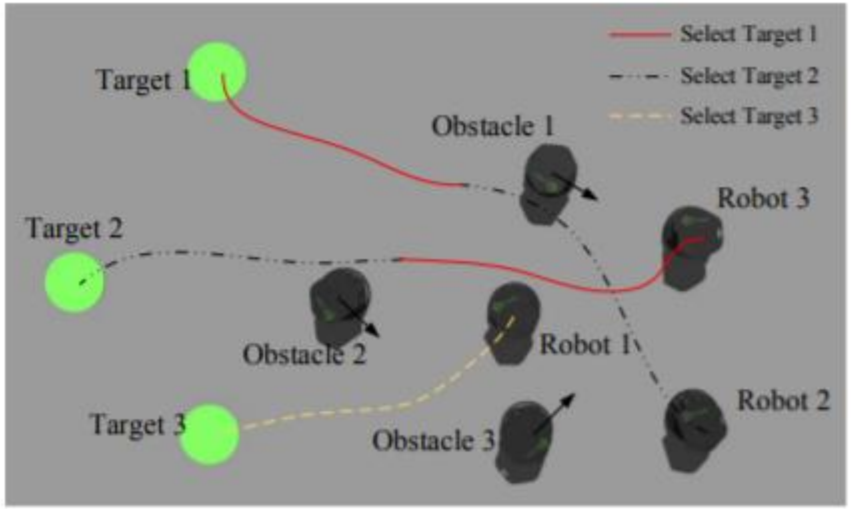
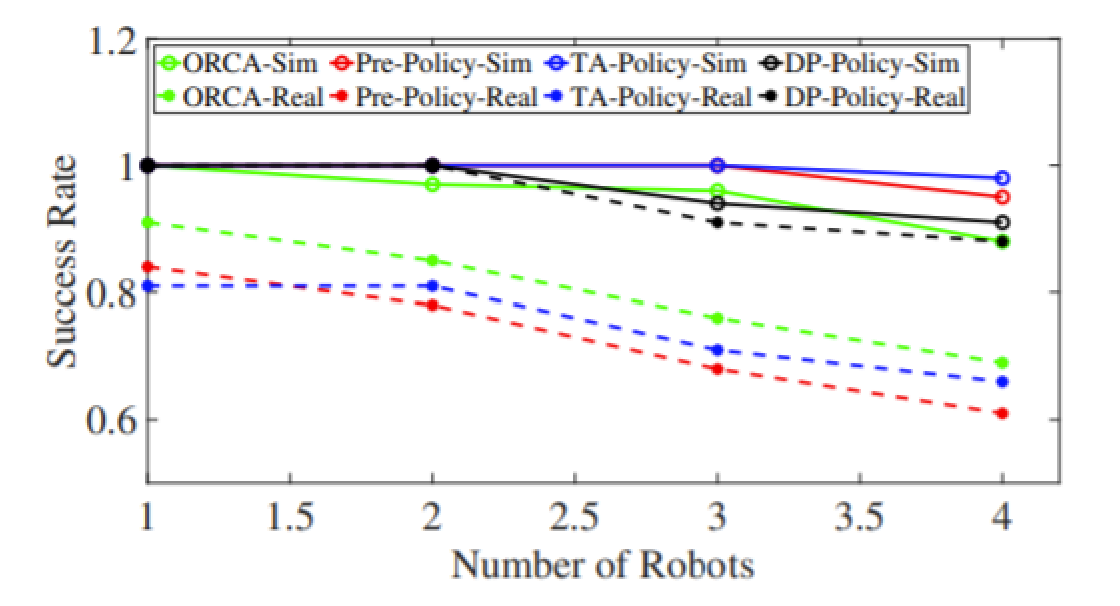
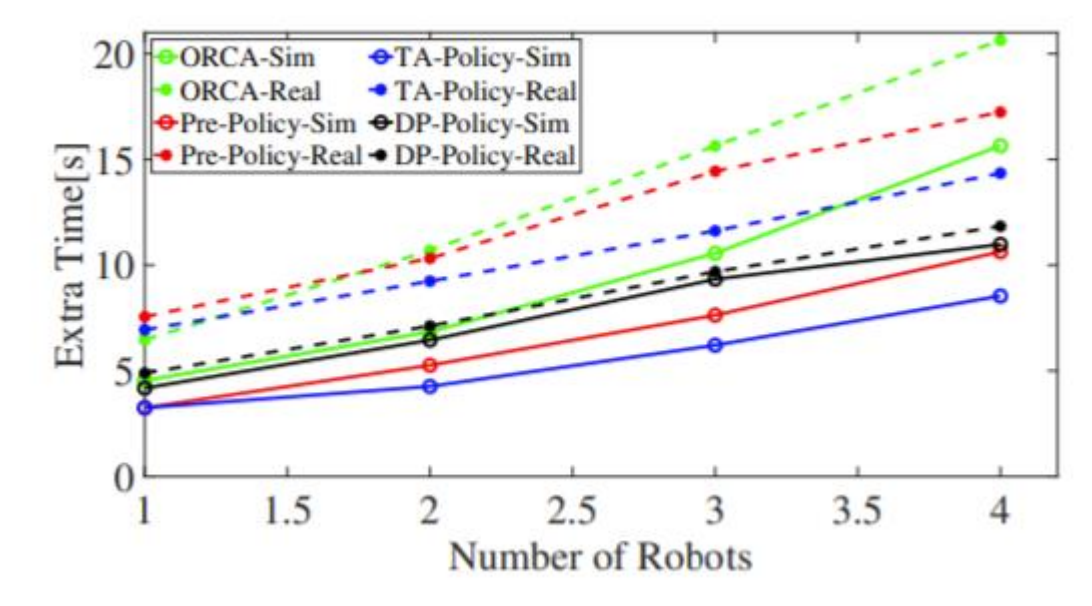


Fig. 4. Simulation scenario

Results

- Baseline – ORCA (computes “allowed” velocities as to not collide)
- Metrics
  - Success rate – # of bots that arrive to the goal
  - Extra time – (avg bot travel time - lower bound on travel)





Summary

- Introduced a **cooperative scheme** for multi-robot navigation
- Collision avoidance
- Closest **target** location **allocation**
- **Dynamics model** for real world transfer
- **Actor-critic** based neural network model
- Two experiments – simulation & real world

Thank you for your attention!

Image sources: the paper
DRL basics: Straka’s lectures

What I think about the paper

- The dynamics (noise) is nice
- Clearly written main results
- The model is not too complicated
- All steps could be easily used in other models and scenarios
- ~~The robots are cute~~

- The models with allocation/dynamics are trained longer
- Only one env/robot type, baseline model
- No confidence intervals in plots
- No code ☹
- The permutation allocation is lame