Model-based Reinforcement Learning for Decentralized Multiagent Rendezvous

Rose E. Wang*
Google Research, Stanford University
rewang@stanford.edu

J. Chase Kew
Robotics at Google
jkew@google.com

Dennis Lee
Google Research
ldennis@google.com

Tsang-Wei Edward Lee
Robotics at Google
tsangwei@google.com

Tingnan Zhang
Robotics at Google
tingnan@google.com

Brian Ichter
Robotics at Google
ichter@google.com

Jie Tan
Robotics at Google
jietan@google.com

Aleksandra Faust
Google Research
faust@google.com

Abstract: Collaboration requires agents to align their goals on the fly. Underlying the human ability to align goals with other agents is their ability to predict the intentions of others and actively update their own plans. We propose hierarchical predictive planning (HPP), a model-based reinforcement learning method for decentralized multiagent rendezvous. Starting with pretrained, single-agent point to point navigation policies and using noisy, high-dimensional sensor inputs like lidar, we first learn via self-supervision motion predictions of all agents on the team. Next, HPP uses the prediction models to propose and evaluate navigation subgoals for completing the rendezvous task without explicit communication among agents. We evaluate HPP in a suite of unseen environments, with increasing complexity and numbers of obstacles. We show that HPP outperforms alternative reinforcement learning, path planning, and heuristic-based baselines on challenging, unseen environments. Experiments in the real world demonstrate successful transfer of the prediction models from sim to real world without any additional fine-tuning. Altogether, HPP removes the need for a centralized operator in multiagent systems by combining model-based RL and inference methods, enabling agents to dynamically align plans.[2]

Keywords: multiagent systems; model-based reinforcement learning

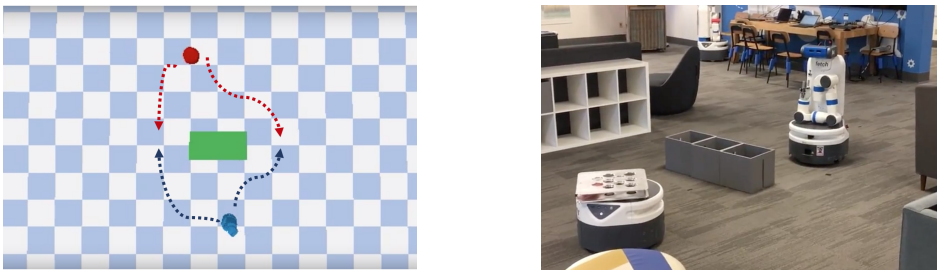
1 Introduction

Imagine you and your friend plan to meet up, and you find yourselves at two ends of a busy crosswalk. How do you efficiently meet at a common location? There are several possibilities that come to mind and might be efficient ways of accomplishing this: Either you could wait for your friend to join you on your side (or vice versa) or you both can attempt to meet in the middle. But what about the people in your way? You attempt to weave around the group by turning left, but you see your friend heading in the opposite direction. Without communication, you understand that both of you imagined different locations and so you replan your route to meet them more quickly.

The above example is the focus of this paper and an instance of a decentralized *rendezvous task* [1, 2], where agents must align their goals [3, 4, 5] without explicit communication. The rendezvous task plays an important role in real world multiagent and human-robot settings for e.g. performing object handovers [6, 7]. While other works address reliable sensor-informed goal navigation [8, 9],

*Corresponding author. The research was conducted during Rose's internship at Robotics at Google.
[2]The video is available at: https://youtu.be/-ydXHUtPzWE

4th Conference on Robot Learning (CoRL 2020), Cambridge MA, USA.



(a) (b)

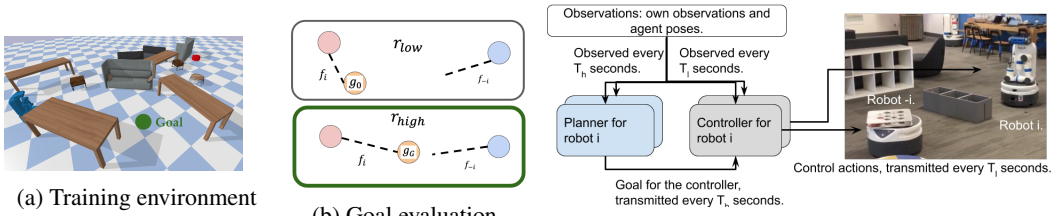
Figure 1: **(a)** Top down view of two independently controlled robots (top, bottom) separated by obstacles (center) must meet each other. How should they move in order to meet? Example trajectories are illustrated in dashed arrows with the robot's corresponding colors. **(b)** Real-robot experiments performing rendezvous task with zero-shot sim2real transfer.

the problem of aligning goals is nontrivial across independent agents with different beliefs without a centralized planner [10] (which isn't always readily available in the real world). The selection of a good rendezvous point depends on the obstacles in the environment, and on the policies and dynamics of the agents. Take for example Fig. 1a where two agents must meet. Without communication, each agent needs to interpret a stream of high-dimensional, noisy sensor inputs to align goals with other agents, while keeping in mind the navigational obstacles and handling miscoordinations, such as agents heading towards goals in opposite directions. Thus, agents must adaptively coordinate [11] to align goals. The obstacle between the agents might prompt a miscoordination, such as the red agent moving west and the blue agent east. Resolving this miscoordination depends on agents' ability to model others' motions and to adapt to diverging intentions using the limited information.

Assuming a decentralized, real-world multiagent system, there are key questions in modelling, predicting, and accounting for other agents' behavior. These questions are three-fold: How should agents coordinate using high-dimensional and imperfect sensor inputs? How should agents cooperate with others under partial observability and uncertainty? How should agents resolve miscoordinations? Our contribution is a holistic approach to address these challenges. We present a decentralized, learning-based system to solve the rendezvous task. Akin to the standard navigation pipeline, our learning-based system consists of three modules: control, prediction, and planning. A robot's control module is equipped with a pre-trained, imperfect navigation policy that can navigate to a given location in the obstacle-laden environment. Using only an agent's own observations and assumed goal, the prediction module learns via self-supervision to predict the motion of agents. The planning module, hierarchical predictive planning (`HPP`), is a hierarchical model-based reinforcement learning (RL) method. It selects and updates its beliefs over rendezvous points: The prediction module evaluates the points, and the evaluations are used to update an agent's beliefs over potential rendezvous points. The planning module outputs a rendezvous point to the control policy for execution. While the hierarchical planning and control setup are not unusual [12, 13], our work closes the loop between the control and planning for decentralized multiagent systems through the motion predictors. We believe this work is of interest to the larger a) multiagent community as a real-world example of a decentralized, cooperative task using noisy sensors and imperfect controllers, b) motion planning community as an example of a learning-based planning system that closes the loop between the planner and controller, and c) RL community as an example of model-based RL as feedback in a hierarchical, self-supervised prediction setting.

2 Related works

Self-supervised prediction and model-based RL methods have also helped make planning more sample-efficient [14, 15] and more interpretable [16, 17]. Additionally, prior works have used learned prediction models for goal proposal and selection [18, 19]. However, these learning methods do not work in the multiagent setting, where reasoning about other agents is key. There are works that learn motion predictive models from a third party point of view [20, 21, 22], however these works either cannot scale to real-world sensors or cannot be transferred from simulation to reality. There are also several prior works in human motion prediction [23] applied to the collision-avoidance task [24, 25], another example of a non-communicating multiagent navigation problem. However, the rendezvous task presents a different navigation challenge where methods for collision-



(a) Training environment

(b) Goal evaluation.

(c) HPP architecture.

Figure 2: **(a)** Motion prediction model, $\boldsymbol{f}_i, \boldsymbol{f}_{-i}$, training environment with randomly filled obstacles. All agents (left, upper right) are given the same random goal (center) and move with their own P2P policies towards it. **(b)** Goal ($\boldsymbol{g}_0$ and $\boldsymbol{g}_G$) evaluation during the high-level policy $\boldsymbol{\Pi}$ execution. At the end of a simulated trajectory, the agents (left and right) are either a) far or b) close to each other. $\boldsymbol{g}_G$ is a better goal than $\boldsymbol{g}_0$ because agents end up closer to each other. **(c)** HPP architecture at run-time. Each agent has a planner and P2P controller running at different frequencies. Each agent receives its own sensor information and agent poses. The planner proposes a goal from the agent's beliefs about how to best align goals among agents. Using the goal, the controller outputs an action for the agent to perform.

avoidance do not apply: it requires agents to coordinate both in time and space to align goals. There has been work to learn coordination end-to-end via the paradigm of centralized training and decentralized execution [26, 27]. However, as we show in experiments, this class of methods fails to scale in real-world, more challenging environments. Our work is most similar to [3] which studies decentralized multiagent systems and aligning team goals by dynamically learning about the other agents via inference and model-based planning. However, similar to other decentralized multiagent planning methods [28, 29], they make assumptions such as having observations of the entire state of the grid world environment (a bird's eye view), access to a perfect model of agents and object interactions (i.e. their model of the world is not learned) and ground truth knowledge of which objects cause collision. In order to bring our robots closer to the real world, our work does not make these assumptions, making the rendezvous task and aligning goals across agents more challenging.

3 Problem definition and preliminaries

We define the problem space as a decentralized partially observable Markov Decision Process (Dec-POMDP) [30]. The Dec-POMDP, $\mathcal{M}$, is the tuple $\langle n, \mathcal{S}, \mathcal{O}, \mathcal{A}_{1\ldots n}, T, \mathcal{R}, \gamma \rangle$. n is the number of agents. We adopt the game theory notation i and $-i$, which refer to agent i and all agents except i respectively. $\mathcal{S}$ is a set of states describing the possible configurations for agents and objects in the environment. We assume the true state space to be hidden and we do not explicitly model it. The joint observation space is $\mathcal{O} = [\mathcal{O}_1, \ldots, \mathcal{O}_n]$, where $\mathcal{O}_i$ is agent i's observations. $\mathcal{A}_{1\ldots n}$ is the joint action space with $\boldsymbol{a}_i \in \mathcal{A}_i$ being the set of actions available to agent i. The transition function $T : \mathcal{S} \times \mathcal{A}_{1\ldots n} \times \mathcal{S} \rightarrow [0, 1]$ is the probability of transitioning from one state to another after each agent takes its respective action $a_{1\ldots n}$. As with the true system state, we assume the transition function is unknown. The reward $\mathcal{R}$ maps system state and action to a scalar that represents the collective reward function. Finally, $0 < \gamma < 1$ is the discount factor.

Each robot observes noisy approximations of agent poses, its own sensor observations, and relative position of its goal (selected by its own planner) in polar coordinates, yielding $\mathcal{O}_i = [\boldsymbol{p}_i, \boldsymbol{p}_{-i}, \boldsymbol{o}, \boldsymbol{g}] \in \mathcal{O}_i$. Each pose consists of the relative x (meters), relative y (meters) and relative heading (radians) information of the agent. Because our work focuses on dynamically aligning navigation goals of independent agents with prediction and planning, we leave the problem of inferring poses of other agents from local observations for future work. The actions for each robot are the robot's linear and angular velocities $\boldsymbol{a}_i = [v, \theta]$.

We assume that each agent is capable of single-agent navigation in obstacle-laden environments, using a policy which we refer to as a P2P (point-to-point) policy. These policies are pre-trained in a separate set of environments, and used here as building blocks without additional training.

4 Methods

First, the method trains motion prediction models in simulation for the given P2P control policies. This training is done in a set of environments independent from the deployment environments. Section 4.1 describes the prediction model training.

Second, after the prediction models are trained, a model-based reinforcement learning planner uses the learned models in the deployment environments to guide the agents towards the rendezvous. Each agent runs its own planner and P2P controller (Figure 2c). The planner takes the agent's observations and outputs a goal for the controller. The controller takes the planner's goal and agent's sensor observations and outputs a control action to the robot. When the planner selects the next goal, it takes into account what it believes the other agents would do if they shared the goal of completing the rendezvous task. To perform this reasoning, the planner uses prediction models to predict trajectories. These trajectories are used to evaluate goals against the joint team objective. Section 4.2 outlines the algorithm. Together, the model and model-based RL planning enable real-time adjustments, recover on-line from miscoordinations, and lead agents to task completion in unseen environments.

4.1 Motion prediction model training via self-supervision

The prediction models predict an agent's goal-conditioned motion. Similar to [31, 32], we learn two models, *self-prediction* ($\boldsymbol{f}_i$) and *other-prediction* ($\boldsymbol{f}_{-i}$), both of which are trained on observations of an agent following a hidden controller. The key difficulty lies in training $\boldsymbol{f}_{-i}$ without access to all of agent $-i$'s sensor observations, particularly lidar. The model must learn to predict the other agent's likely motion based on its own lidar readings. The self-prediction model takes in a history length h of the agent's own sensors $\boldsymbol{p}_i^{t-h:t}, \boldsymbol{o}_i^{t-h:t}$ and fixed training goal $\boldsymbol{g}$ between time steps $t-h$ to t. The self-prediction model outputs $\Delta\boldsymbol{p}_i^{t+1}, \Delta\boldsymbol{o}_i^{t+1}$, i.e. the difference between its next and current pose and sensors with respect to its own policy

$$(\Delta\boldsymbol{p}_i^{t+1}, \Delta\boldsymbol{o}_i^{t+1}) = \boldsymbol{f}_i(\boldsymbol{p}_i^{t-h:t}, \boldsymbol{o}_i^{t-h:t}, \boldsymbol{g}). \tag{1}$$

Similarly, the other-prediction model, which models the motion of agent $-i$, takes in a history length h of agent $-i$'s approximated pose, agent i's own sensors and the fixed training goal. It outputs the change in agent $-i$'s pose and lidar with respect to agent $-i$'s controller,

$$(\Delta\boldsymbol{p}_{-i}^{t+1}, \Delta\boldsymbol{o}_i^{t+1}) = \boldsymbol{f}_{-i}(\boldsymbol{p}_{-i}^{t-h:t}, \boldsymbol{o}_i^{t-h:t}, \boldsymbol{g}). \tag{2}$$

Note that the models do not store actions because they learn anticipated poses of agents conditioned on goals. This approach avoids having to know the exact action spaces of robots.

Data collection Given the P2P controllers $\boldsymbol{\pi}_i$, $i = 1 \ldots n$ and a goal, we collect examples of agents in the environment executing their $\boldsymbol{\pi}_i$'s to move to the goal. To provide diverse experiences, we randomize the obstacles in our multiagent simulation environment (see Figure 2a for an example). For each agent, we collect two datasets $\mathcal{D}_i$ (for $\boldsymbol{f}_i$) and $\mathcal{D}_{-i}$ (for $\boldsymbol{f}_{-i}$) which contain the training information for the prediction models explained previously.

$$\mathcal{D}_i = [(\boldsymbol{p}_i^{t-h:t}, \boldsymbol{o}_i^{t-h:t}, \boldsymbol{g}), (\Delta\boldsymbol{p}_i^{t+1}, \Delta\boldsymbol{o}_i^{t+1})]$$
$$\mathcal{D}_{-i} = [(\boldsymbol{p}_{-i}^{t-h:t}, \boldsymbol{o}_i^{t-h:t}, \boldsymbol{g}), (\Delta\boldsymbol{p}_{-i}^{t+1}, \Delta\boldsymbol{o}_i^{t+1})].$$

Model training $\mathcal{D}_i$ and $\mathcal{D}_{-i}$ are used to train the *self-prediction* ($\boldsymbol{f}_i$) and *other-prediction* ($\boldsymbol{f}_{-i}$) models. Both predictors are approximated with deep neural networks consisting of four fully connected layers, trained with mean squared loss.

4.2 HPP planner

The decentralized policy $\boldsymbol{\Pi}_i$ uses 1 $\boldsymbol{f}_i$ and $(n-1)$ $\boldsymbol{f}_{-i}$ prediction models to plan and evaluate goals. We use the cross-entropy method (CEM) [33] to convert goal evaluations into belief updates over potential rendezvous points. Algorithm 1 describes $\boldsymbol{\Pi}_i$ that runs on an agent i. The intuition behind $\boldsymbol{\Pi}_i$ is that each agent, independently, simulates a fictitious centralized agent that fixes the goal of all agents (Lines 4-8). The goal pre-conditions the motion predicted by $\boldsymbol{f}_i$ and $\boldsymbol{f}_{-i}$. Conditioned on a

Algorithm 1 HPP 's high-level policy, Π_i.

Input: Pretrained prediction models $\boldsymbol{f}_i, \boldsymbol{f}_{-i}$; Observations $\mathcal{O}_i = (\boldsymbol{p}_i, \boldsymbol{p}_{-i}, \boldsymbol{o}_i, \boldsymbol{g}_i)$
Output: Agent's new goal $\hat{\boldsymbol{g}}_i$
1: Initialize goal distribution $\mathcal{N}(\boldsymbol{p}_\mu, \boldsymbol{p}_\sigma^2)$ using agent poses $\boldsymbol{p}_i, \boldsymbol{p}_{-i}$
2: **for** $l <$ MaxIterations **if** $\boldsymbol{p}_\sigma > \epsilon$ **do**
3: Sample N goals G from current distribution
4: **for all** $\boldsymbol{g}_j \in G$ **do**
5: Initialize predicted poses $\hat{\boldsymbol{p}}_i, \hat{\boldsymbol{p}}_{-i}$ & sensors $\hat{\boldsymbol{o}}_i^{(s)}, \tilde{\boldsymbol{o}}^{(s)}$ with observation $\mathcal{O}_i$
6: **for** k=1…T **do**
7: Predict changes in pose $\Delta\boldsymbol{p}_i, \Delta\boldsymbol{p}_{-i}$ & sensors $\Delta\boldsymbol{o}, \Delta\tilde{\boldsymbol{o}}^{(s)}$ with Eq. (1) and (2)
8: Update $\hat{\boldsymbol{p}}_i, \hat{\boldsymbol{p}}_{-i}, \hat{\boldsymbol{o}}_i^{(s)}, \tilde{\boldsymbol{o}}^{(s)}$
9: **end for**
10: Compute reward for $\boldsymbol{g}_j = \mathcal{R}(\hat{\boldsymbol{p}}_i, \hat{\boldsymbol{p}}_{-i})$ using Eq. (3)
11: **end for**
12: Select M goals with highest rewards and update $\mathcal{N}(\boldsymbol{p}_\mu, \boldsymbol{p}_\sigma^2)$
13: **end for**
14: **return** $\boldsymbol{p}_\mu$

proposed goal, the algorithm predicts the poses of agents for a horizon of T time steps in the future (Lines 6-9). The poses are generated from sequential roll outs of the prediction models $\boldsymbol{f}_i$ and $\boldsymbol{f}_{-i}$. Each goal is then evaluated by scoring the anticipated system state using the rendezvous task reward shown in Eq. (3) (Line 10). The reward is the negative difference in agent positions, and is 0 if the task is completed, which happens when agents meet within a predetermined distance, d, from each other. Specifically, the reward is defined as

$$\mathcal{R}(\boldsymbol{p}_{1..n}) = \begin{cases} 0 & |\boldsymbol{p}_j - \boldsymbol{p}_\mu| < d \quad \forall j \in 1..n \\ \sum_{j,k\neq j} -|\boldsymbol{p}_k - \boldsymbol{p}_j| & otherwise \end{cases} \text{ where } \boldsymbol{p}_\mu = \frac{1}{n}\sum_{k\in 1..n} \boldsymbol{p}_k \qquad (3)$$

is the center point of the robots' poses (average of their locations), d controls the precision of the rendezvous, n is the number of agents. See Appendix A for more details.

The planner uses the rewards to update the distribution over goals to favor ones that bring agents closer together (Line 12). Figure 2b illustrates this process: $\boldsymbol{\Pi}_i$ evaluates the different goals for each agent by checking the reward it expects to receive after a time horizon given each goal, and it assigns higher reward to the goal that closes the distance between agents.

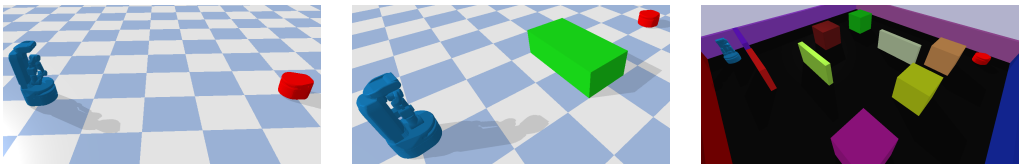
Finally, to run the HPP and complete a coordinated rendezvous task without coordination, each agent i runs an instance of $\boldsymbol{\Pi}_i$. The input to $\boldsymbol{\Pi}_i$ is n prediction models: one self-prediction and $n-1$ other agents prediction models, as each agent might be running a different P2P policy. Every T_h timesteps, $\boldsymbol{\Pi}_i$ observes its sensors, receives poses of all agents, and outputs a recommended goal for agent i; in other words, T_h is the frequency of high-level planner, which controls how frequently the goal is updated). P2P policy receives the goal and drives the agent to the goal, performing actions every $T_l < T_h$ seconds; in other words, T_l is frequency of low-level controller. The process stops when the agents are close to each other or time runs out.

5 Results

To evaluate that presented model we answer the following questions: 1) Does HPP leverage the prediction models to align goals? 2) Does HPP effectively reduce the rendezvous time? 3) Does HPP generalize and effectively handle the rendezvous task in real environments with zero-shot transfer? But first, we describe the experiment setup, baselines and environments.

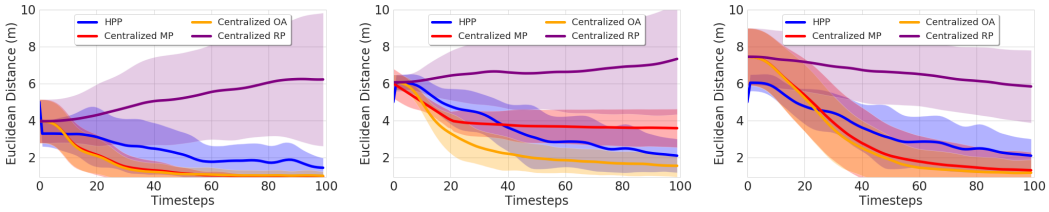
Setup We use Fetch and Freight mobile robots [34]. The observations are 2D lidar with 222 rays. The planning parameters for HPP are $T = 5$, $T_l = 1$, $T_h = 10$, which were selected based on the ablation studies. The ϵ convergence criteria for CEM is 0.001. Appendix B.1 contrain more details.

Baselines We compare HPP to learned, planning, and centralized baselines. MADDPG [26] is the learned baseline because it is one of the most popular model-free multiagent RL algorithms (see Appendix B.2 for the training details). RRT [35] is the planning baseline. We combine RRT with



(a) Simple, no obstacle world. (b) Wall world. (c) Hard navigation world.

Figure 3: Simulated testing environments for the rendezvous task. (a) is a simple environment with no obstacles. (b) is an environment with a wall in between the agents. This environment tests whether agents can converge to one side of the wall (left or right) and thus break the symmetry in the goal alignment problem, a well known multiagent challenge [36, 37, 38]. (c) is a challenging navigation environment with many obstacles in the way of rendezvous.



(a) Simple world evaluation. (b) Wall world evaluation. (c) Navigation world evaluation.

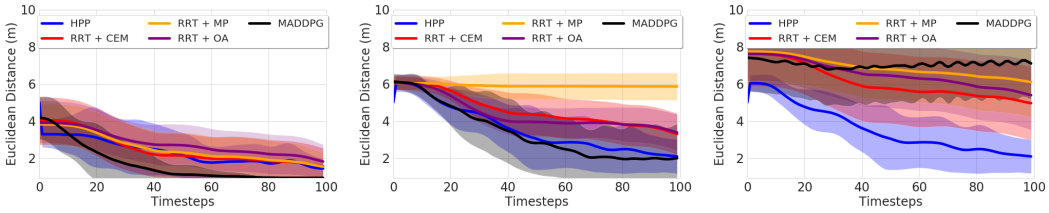
Figure 4: Distance between agents (lower is better) over planning time in unseen simulation environments (Figure 3). `HPP` (blue) vs centralized baselines.

CEM (`RRT+CEM`), which simulates the motion of agents using `RRT` and selects goals using `CEM` as described in Section 4.2. Lastly, the centralized baselines rely on heuristic to fix the goals of agents to a common location – agents do not perform inference or maintain uncertainty of where to go like `HPP` agents do. Specifically, the centralized baselines set the high level planner's goal to the midpoint of the agents (`Centralized MP`), the other agent's position (`Centralized OA`) or a random point in the environment (`Centralized RP`). For example, `Centralized MP` and `RRT+MP` means that no goal inference is performed and agents move towards the midpoint of their positions using our low-level policy or `RRT` respectively.

Environments We use three sets of environments. First, we train the motion prediction models in simulation in a cluttered environment depicted in Figure 2a. The training details and learning results are in the Appendix B.3. Next, we evaluate `HPP` in two sets of separate environments: three simulation (Figure 3) and three real environments (Figure 6). The evaluation environments are designed with different difficulty levels. Due to the COVID-19 shutdowns, we were unable to run all the baselines in the real world. Therefore, we constructed high-fidelity scans (Appendix D) and evaluated all baselines in these environments. We report both results.

5.1 Leveraging prediction models to align goals

Figure 4 demonstrates that `HPP` effectively leverages prediction models for goal alignment. It compares `HPP` to centralized heuristic baselines in the simulated test environments (Figure 3). First, `HPP` brings agents closer together than the fixed random goal baseline `Centralized RP`, illustrating that goals must be both aligned across agents and set intelligently (e.g. set using prediction models or an improved heuristic) for agents to align goals efficiently. Second, `HPP` leverages its prediction model to actively select better goals than `Centralized MP` in the wall environment, where `Centralized MP` is unable to overcome the wall obstacle. Furthermore, `HPP` leverages its prediction model to actively align agent trajectories for the rendezvous, unlike `Centralized OA`. Visual inspection of their video performance shows that `Centralized OA` agents end up **chasing each other** in tight circles, without stopping, because they're unable to predict where the other agent might be in the future. Although they end up closer together, they do not successfully rendezvous.



(a) Simple world evaluation. (b) Wall world evaluation. (c) Navigation world evaluation.

Figure 5: Distance between agents (lower is better) over planning time in unseen simulation environments (Figure 3). HPP (blue) vs. RRT, RRT using heuristics and MADDPG.

(a) Simple world. (b) Wall world. (c) Table forest world. (d) Roundabout world.

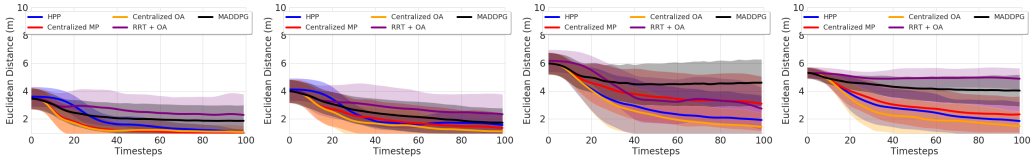
Figure 6: Real-world environments used for evaluating HPP and MADDPG prior to COVID-19. These environments are also reconstructed from high-fidelity scans in order to compare performances of other baselines.

5.2 Reducing rendezvous time

Figure 5 shows that HPP reduces the rendezvous time more than the decentralized baselines—RRT+CEM and MADDPG—and RRT using centralized heuristics. In RRT+OA, each agent builds and traverses an RRT from its position to the other agent's position. The RRT is recalculated every 40 timesteps. RRT+MP is similar, but the goals of the RRTs are the midpoint between the two agents. HPP performs similarly to RRT alternatives in the simple environment, and outperforms the baselines in all other environments. Similar to Centralized MP, all three RRT baselines are unable to overcome the symmetry challenge; RRT+MP fails because the RRT planner fails to plan around the wall, RRT+OA fails because the RRT paths are inefficient to overcome the wall obstacle. RRT+CEM performs slightly better than other RRT alternatives in the navigation environment—we hypothesize CEM facilitates aligning the goals of agents. Nonetheless, RRT+CEM performs much worse than HPP because the RRT causes frequent backtracking and zigzagging. This inefficient motion makes it difficult for the CEM to distinguish a good goal from a bad based on the agents' early progress toward it. These failure modes persist in the navigation environment, a more challenging domain.

Finally, we see that MADDPG outperforms HPP in the simple world. MADDPG performs similarly HPP in the wall world and completely fails in the hard navigation world, where agents only turn in circles. We found it surprising that MADDPG was able to generalize to any testing environments because we expected a model-free multiagent algorithm to require fine-tuning in unseen environments. We hypothesize MADDPG learns to use agent poses and ignores lidar observations. This would explain why it performs poorly in obstacle-laden environments where lidar observations are key in navigation, yet performs similarly to our method in simple, obstacle-minimal environments.

5.3 Generalization to real environments



(a) Simple world. (b) Wall world. (c) Table forest world. (d) Roundabout world.

Figure 7: Distance between agents (lower is better) over planning time in unseen testing environments reconstructed from high-fidelity scans of the real world. HPP (blue) vs. RRT+OA, centralized heuristics and MADDPG.

Finally, Figure 7 shows that HPP generalizes to real environments with zero-shot transfer. Appendix D includes the video link to these experiments, images of the reconstructed environments, and some trajectories of the real-world experiments with HPP .

The results are consistent with the previous section: in the simple environment, all the baselines perform similarly to each other; in the wall environment, Centralized MP struggles to overcome the symmetry challenge from the wall object; and finally, as environments become more challenging, MADDPG struggles to scale and utilize its lidar observations to navigate, resulting in agents roaming in circles and unsuccessfully converging to a rendezvous location. As before, Centralized OA agents closely chase each other whereas HPP complete the task and handle this type of miscoordination. We hypothesize the reason that HPP struggles on the table top world is due to the narrow navigation spaces in contrast to the open spaces in other environments; the narrowness discourages agents from meeting and aligning goals during their prediction rollouts without colliding into surrounded objects.

5.4 Environment difficulty and hyperparameters

Increasing the planning decision making frequency and planning horizon yield disproportionately larger performance improvements in the more complex environments, potentially justifying increased computational cost. We recommend tuning the decision making rate once per environment. See Appendix C.1 for details.

Ablation studies (Appendix C.2) show that learning to predict change in pose relative to the agent that makes predictions yields better prediction models than learning to predict an absolute pose or change in pose in the global coordinate system. This is reasonable because the predictions are made from the vantage point of the agent making the decisions.

5.5 Discussion, future work, and broader impact

While this work focuses on goal alignment in the context of the navigation rendezvous task, HPP can be used for other **decentralized multiagent coordination tasks without explicit coordination** in the future, by applying a different *task objective*, *agent's primitive skills*, and *team size*. First, we assume that all agents share a *task objective*, specified with a reward function in Eq. (3). Substituting a different task reward will result in learning the task defined with the given reward function. Second, this work builds a modular system out of learned components, by assuming pretrained P2P navigation policies are given. Since HPP and motion model predictions do not make assumptions about the nature of policies, P2P navigation policies can be substituted with task-appropriate *skill primitives* including *heterogeneous teams of robots*. Lastly, the algorithm makes no assumptions about the *team size*, and future work can consider larger teams.

6 Conclusions

This work presents a model-based RL approach to solving a decentralized rendezvous task. First, the method learns to approximate own and teammates' motion models that reflect robots' abilities and limitations with respect to their control policies via self-supervised learning; the motion models decouple the capabilities of the agents from the task. Next, the method uses a planning module that iteratively proposes subgoals for the agent to move towards. HPP maintains a belief distribution over goals using a joint objective function and the learned motion models. Finally, HPP updates this distribution and replans using motion predictions to address misaligned agent goals, i.e. miscoordination. We demonstrate the generalization of the motion predictor models to new environments, both in simulation and in the real world. Compared to the baselines, without explicit coordination between the agents, HPP is more likely to complete the rendezvous task by selecting goals that are feasible, coordinated, and unobstructed.

Acknowledgments

We thank Michael Everett, Oscar Ramirez and Igor Mordatch for the insightful discussions.

References

[1] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[2] A. Faust, N. Malone, and L. Tapia. Preference-balancing motion planning under stochastic disturbances. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 3555–3562, 2015. doi: 10.1109/ICRA.2015.7139692.

[3] R. E. Wang, S. A. Wu, J. A. Evans, J. B. Tenenbaum, D. C. Parkes, and M. Kleiman-Weiner. Too many cooks: Coordinating multi-agent collaboration through inverse planning. *arXiv preprint arXiv:2003.11778*, 2020.

[4] K. Tumer and M. Knudson. Aligning agent objectives for learning and coordination in multi-agent systems.

[5] K. Tumer, A. K. Agogino, and D. H. Wolpert. Learning sequences of actions in collectives of autonomous agents. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems: part 1*, pages 378–385, 2002.

[6] K. Strabala, M. K. Lee, A. Dragan, J. Forlizzi, S. S. Srinivasa, M. Cakmak, and V. Micelli. Toward seamless human-robot handovers. *Journal of Human-Robot Interaction*, 2(1):112–132, 2013.

[7] C.-M. Huang, M. Cakmak, and B. Mutlu. Adaptive coordination strategies for human-robot handovers. In *Robotics: science and systems*, volume 11. Rome, Italy, 2015.

[8] L. Chiang, A. Faust, M. Fiser, and A. Francis. Learning navigation behaviors end-to-end with autorl. 4(2):2007–2014, April 2019. ISSN 2377-3766. doi:10.1109/LRA.2019.2899918.

[9] T. Fan, X. Cheng, J. Pan, P. Long, W. Liu, R. Yang, and D. Manocha. Getting robots unfrozen and unlost in dense pedestrian crowds. *IEEE Robotics and Automation Letters (RA-L)*, 2019. URL http://arxiv.org/abs/1810.00352.

[10] L. Brunet, H.-L. Choi, and J. How. Consensus-based auction approaches for decentralized task assignment. In *AIAA guidance, navigation and control conference and exhibit*, page 6839, 2008.

[11] P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[12] A. Wahid, A. Toshev, M. Fiser, and T. E. Lee. Long range neural navigation policies for the real world. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 82–89, 2019.

[13] A. Faust, O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson, and L. Tapia. Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, pages 5113–5120, Brisbane, Australia, 2018.

[14] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *arXiv preprint arXiv:1812.00568*, 2018.

[15] N. Hirose, F. Xia, R. Martín-Martín, A. Sadeghian, and S. Savarese. Deep visual mpc-policy learning for navigation. *IEEE Robotics and Automation Letters*, 4(4):3184–3191, 2019.

[16] A. S. Polydoros and L. Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

[17] J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion, 2018.

[18] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust. Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies. *IEEE Robotics and Automation Letters*, 4 (4):4298–4305, 2019.

[19] S. Bansal, V. Tolani, S. Gupta, J. Malik, and C. Tomlin. Combining optimal control and learning for visual navigation in novel environments. *arXiv preprint arXiv:1903.02531*, 2019.

[20] N. C. Rabinowitz, F. Perbet, H. F. Song, C. Zhang, S. Eslami, and M. Botvinick. Machine theory of mind. *arXiv preprint arXiv:1802.07740*, 2018.

[21] N. Jaques, A. Lazaridou, E. Hughes, C. Gulcehre, P. Ortega, D. Strouse, J. Z. Leibo, and N. De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International Conference on Machine Learning*, pages 3040–3049, 2019.

[22] S. Barrett, P. Stone, S. Kraus, and A. Rosenfeld. Learning teammate models for ad hoc teamwork. In *AAMAS Adaptive Learning Agents (ALA) Workshop*, pages 57–63, 2012.

[23] S.-Y. Chung and H.-P. Huang. Predictive navigation by understanding human motion patterns. *International Journal of Advanced Robotic Systems*, 8(1):3, 2011.

[24] M. Everett, Y. F. Chen, and J. P. How. Collision avoidance in pedestrian-rich environments with deep reinforcement learning. *arXiv preprint arXiv:1910.11689*, 2019.

[25] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard. Feature-based prediction of trajectories for socially compliant navigation. In *Robotics: science and systems*, 2012.

[26] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments, 2017.

[27] R. E. Wang, M. Everett, and J. P. How. R-maddpg for partially observable environments and limited communication. 2019.

[28] D. Claes, F. Oliehoek, H. Baier, K. Tuyls, et al. Decentralised online planning for multi-robot warehouse commissioning. In *AAMAS'17: PROCEEDINGS OF THE 16TH INTERNATIONAL CONFERENCE ON AUTONOMOUS AGENTS AND MULTIAGENT SYSTEMS*, pages 492–500, 2017.

[29] V. R. Desaraju and J. P. How. Decentralized path planning for multi-agent teams in complex environments using rapidly-exploring random trees. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4956–4961. IEEE, 2011.

[30] F. A. Oliehoek. Decentralized pomdps. In *Reinforcement Learning*, pages 471–503. Springer, 2012.

[31] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7559–7566. IEEE, 2018.

[32] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

[33] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[34] D. K. E. D. Melonee Wise, Michael Ferguson and D. Dymesich. Fetch & freight: Standard platforms for service robot applications. In *Workshop on Autonomous Mobile Service Robots held at the 2016 International Joint Conference on Artificial Intelligence*, 2016.

[35] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Dept., Iowa State University, Oct. 1998.

[36] J. Li, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig. Symmetry-breaking constraints for grid-based multi-agent path finding. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6087–6095, 2019.

[37] J. Li, G. Gange, D. Harabor, P. J. Stuckey, H. Ma, and S. Koenig. New techniques for pairwise symmetry breaking in multi-agent path finding. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 193–201, 2020.

[38] A. Chapman and M. Mesbahi. On symmetry and controllability of multi-agent systems. In *53rd IEEE Conference on Decision and Control*, pages 625–630. IEEE, 2014.

[39] M. Wise, M. Ferguson, D. King, E. Diehr, and D. Dymesich. Fetch and freight: Standard platforms for service robot applications.

[40] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

Appendix

A Discussion on the *rendezvous* task reward

Our work uses a reward function that captures the negative distance between agents' final simulated positions,

$$\mathcal{R}(\boldsymbol{p}_{1..n}) = \begin{cases} 0 & |\boldsymbol{p}_j - \boldsymbol{p}_\mu| < d \quad \forall j \in 1..n \\ \sum_{j,k \neq j} -|\boldsymbol{p}_k - \boldsymbol{p}_j| & otherwise \end{cases} \tag{4}$$

where $\boldsymbol{p}_\mu = \frac{1}{n} \sum_{k \in 1..n} \boldsymbol{p}_k$ is the center point of the robots' positions (average of their locations), d controls the precision of the rendezvous, n is the number of agents.

A previous reward function we used was the negative accumulated distance over a simulated trajectory. However, this assumes agents to strictly close the distance between themselves at every step. In experimentation, we realized agents cannot strictly minimize the distance at every step and selected suboptimal goals with the previous reward function. For example, in obstacle-filled environments, agents need to first navigate around obstacles before rendezvousing. The current reward is designed based on this insight: agents should only care about their final positions, not intermediate ones.

B Evaluation setup

B.1 Robot and planning parameter setup

We use two robots, a Fetch and a Freight [39]. Each observes a 222 beam 2D lidar with 220 degree field of view to match the real robot sensors, as well as the global poses of all agents. Each agent's action space is linear and angular base velocity, clipped to the ranges $[0, 1]$ meters/second and $[-3, 3]$ radians/second respectively. Linear acceleration is limited to $0.4\ m/s^2$ and angular acceleration to 1.48 rad$/s^2$ ($85°/s^2$). In our experiments, the agent's low-level policy operates every timestep (i.e. $T_l = 1$) and the high-level policy operates every 10 timesteps (i.e. $T_h = 10$), unless otherwise specified. The rollout length T is 5. The episodes are 100 T_l timesteps (20 s) long both in training and in evaluation. In the real world experiments, the ROS navigation stack [40] provides the pose observations. As for CEM, $\epsilon = 0.001$, the number of max sampling iterations is 15, the number of samples is 15 and the number of top samples used to update the goal distribution is 5. For the low-level controller, we use a goal-conditioned navigation policy $\boldsymbol{\pi}_i$ trained with [8], although any other P2P policy capable of driving a robot to a goal using local observations would be suitable, such as [9].

B.2 `MADDPG` setup

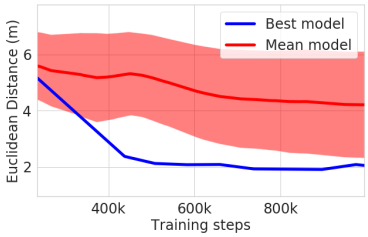


Figure 8: MADDPG training curves. The pink shaded region shows the standard deviation across a batch of training runs. The blue curve is the best trained MADDPG model, which we use as the baseline.

Both the policy and critic networks for `MADDPG` are two layer networks with 64 units in each. Batch size is 1024 and learning rate is 0.0005. `MADDPG` was trained on 10 different seeds in the randomized obstacle environment described in Section 4. `MADDPG` takes 1M steps to converge (Fig. 8). `MADDPG` is also relatively unstable; of 10 policies, 3 failed to converge, while the prediction models are stable to train. In the experiments for Section 5, we used the policy that performed the best (i.e. achieved the highest reward) in the rendezvous task.

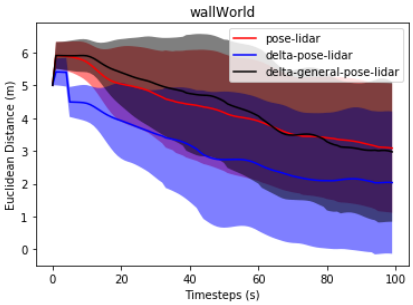
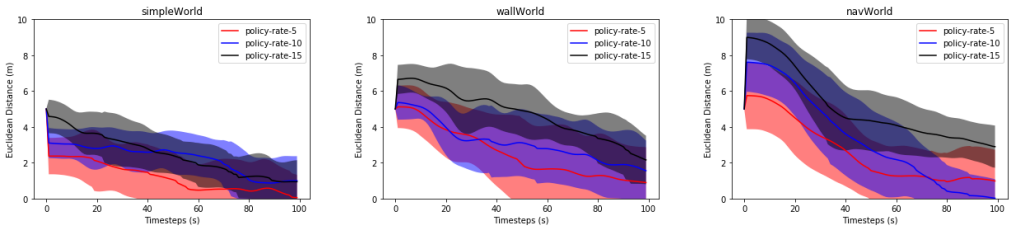


Figure 9: Performance of `HPP` on wall world varying the prediction model. Lower is better. `delta-pose-lidar` (blue) is ours.



(a) Simple world evaluation. (b) Wall world evaluation. (c) Navigation world evaluation.

Figure 10: Planning frequency at which the high level policy Π recalculates new goal across environments. Lower is better. The planning frequency should be tuned per environment.

B.3 Setup for prediction models

Figure 2a shows the training environment for the predictive models $\boldsymbol{f}_i$ and $\boldsymbol{f}_{-i}$ and Figure 3 three previously *unseen* test environments. The simulated evaluations are repeated 10 times where the agents are randomly initialized 5 meters from each other.

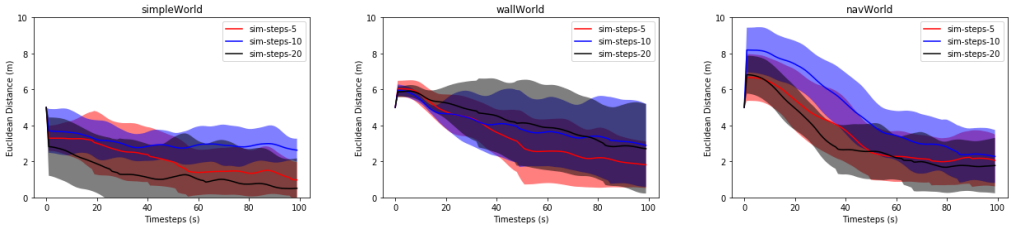
The prediction model is a 4-layer fully connected network with layer units 64, 128, 128 and 64 and a learning rate of 0.001. It is trained on experiences collected from an environment with randomized furniture obstacles (Figure 2a). Goals are randomized within a 20×20 meter square and are not guaranteed to be collision-free. The history trace length, H, is 5 for all agents. Low-level policies navigating towards randomly selected goals collect 50,000 trajectories, or the equivalent of about 11 days of real-time experience. Each prediction model (f_i, f_{-i}) has its own network, trained on 50,000 epochs with batch sizes of 500. The data collection and training take about 3 hours to complete. Both *self-prediction* and *other-prediction* models converge (Figure 12). As expected, *self-prediction* is easier than predicting another agent's motion without having its sensor readings.

C Ablation studies

C.1 Planning hyperparameters

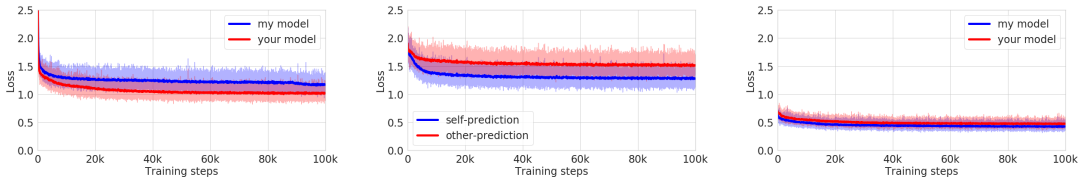
We investigate the role of three planning hyperparameters in Algorithm 1: planning frequency (T_h), planning horizon (T), and goal sampling parameters.

Planning frequency Across the three environments, planning every 5 timesteps leads to faster task completion compared to every 10 or 15 timesteps (Figure 10). Though this is unsurprising, we also note that planning frequency makes more difference in more complex environments. However, frequent planning comes at the cost of higher computational load, so it is helpful to tune it to the desired precision. Figure 10 suggests that the optimal planning rate depends on the environment, so planning rate should be tuned once per environment.



(a) Simple world evaluation. (b) Wall world evaluation. (c) Navigation world evaluation.

Figure 11: Planning horizon evaluations across environments. Lower is better. Planning horizon does not depend on the environment, and should be tuned once per predictor.



(a) Loss for `pose-lidar`. (b) Loss for `delta-pose-lidar`. (c) Loss for `delta-general-pose-lidar`.

Figure 12: Training losses for the different prediction model types. The dark lines are averaged over 100 steps.

Planning horizon Similarly, evidence in Fig. 11 suggests that the performance gap varies based on the environment and the planning horizon. However, the tradeoff of having a longer or shorter horizon is not as significant as varying the planning frequency since an additional rollout is much faster than frequently re-planning and re-evaluating goals, which is what the planning frequency controls.

Goal sampling parameters We iteratively improve the sampled goal using CEM, but any alternative importance sampling technique may work as well. We conducted experiments where we varied the parameters of the CEM: maximum number of iterations and number of top goals (between 5 and 15) used for updating the CEM distribution. None of these ablation experiments indicated that HPP has a strong dependency on the goal sampling parameters.

C.2 Prediction models ablation

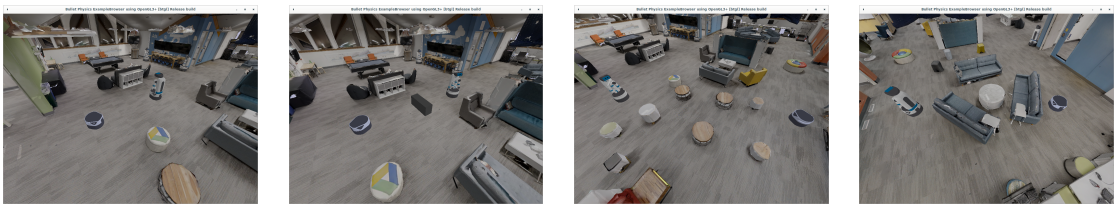
We conduct an ablation study on model representations for $\boldsymbol{f}_i, \boldsymbol{f}_{-i}$. To contrast prediction methods, we train two additional prediction types under the same settings and inputs. We refer to the prediction type presented in Section 4.1 as `delta-pose-lidar`. The two additional ones are `pose-lidar` and `delta-general-pose-lidar`. `pose-lidar` directly outputs the next pose and lidar predictions. `delta-general-pose-lidar` differs from `delta-pose-lidar` in that it predicts the pose of agent $-i$ relative to the current pose of agent i, and predicts the full lidar observation of agent i. Figure 12 shows the training loss for the three prediction types.

We evaluate the task performance of the prediction models on wall world. Fig. 9 shows that `delta-pose-lidar` performs the best while the other two prediction models perform equally well. In particular, the first few timesteps indicate a drastic decrease in euclidean distance since these agents quickly overcome the symmetry-breaking problem presented in wall world. Future work can investigate different prediction schemes.

D Real-world videos

Our work's video linked here[3] shows that evaluation by the euclidean metric does not tell the whole story: The video illustrates how our model-based reinforcement learning is key for agents to ren-

[3] Video link: https://youtu.be/-ydXHUtPzWE



(a) Simple world. (b) Wall world. (c) Table forest world. (d) Roundabout world.

Figure 13: Environments built from scans of the room used for real world evaluations (i.e. Figure 6). These environments were used to evaluate HPP and all baselines.

dezvous. With Centralized OA, agents end up **chasing each other** in tight circles, which reduces the distance between the agents but is not a successful rendezvous. Even though euclidean distance is a standard way to evaluate success rates in robotics and motion planning, the video shows a discrepancy in how the task is completed by the different methods.

Thus, the main takeaways from the video are: HPP outperforms the other baselines and can successfully transfer zero-shot sim-to-real; Centralized OA agents do not rendezvous because they chase each other; RRT+OA is unable to complete the rendezvous in the specified episode time because agents plan zigzagging paths which undo their rendezvous progress; and MADDPG agents rarely rendezvous and more often spin in place because the baseline is unable to transfer into real-world environments and handle the noisy, high-dimensional sensor data.

Some methods are compared in the real world (pre-COVID), and all methods are compared in the simulation environments reconstructed from scans of the same room used in the real world. Images of HPP performing in the real world can be seen in Table 1. The robots are decentralized, which means they plan and execute independently. There is no coordination in their action execution and there is no centralized controller telling the robots where to go.

During the pauses in HPP video evaluations, HPP samples and evaluates goals in real-time before moving towards the goal of its choosing.

The videos also include clips of the agents' goals dynamically changing and an example episode illustrating the training environments for the prediction models. In the video, the waypoints sometimes are placed on top of obstacles. This is because agents select waypoints by whichever brings agents closer. Our algorithm doesn't assume the goals to be reached at the end of a simulated trajectory (Algorithm 1).

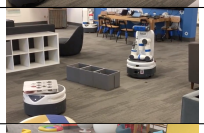
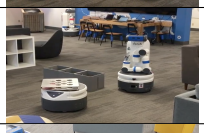
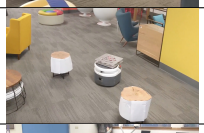
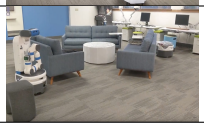
Environment type	1	2	3	4
Simple				
Wall				
Forest				
Roundabout				

Table 1: Images from real world evaluations of HPP on different environment configurations.