Inverse Learning of Robot Behavior for Collaborative Planning

Maulesh Trivedi[1] and Prashant Doshi[2]

***Abstract*— Inverse reinforcement learning (IRL) is an important basis for learning from demonstrations. Observing an agent, human or robotic, perform a task provides information and facilitates learning the task. We show how the agent's preferences learned using IRL can be incorporated in a subject robot's decision making and planning, to enable the robot to spontaneously collaborate with the previously observed agent on the task. We prioritize a real-world application, where a line robot will autonomously collaborate with another robot in sorting ripe and unripe fruit such as oranges. Toward this, our evaluations utilize a colored-ball sorting task as an analog using simulated TurtleBots equipped with Phantom X arms. Our method is comprehensive providing first answers to questions such as how should the robot acquire the complete model for the collaborative planning problem and how should it solve the problem to obtain a plan that permits collaboration without disrupting the line robot's behavior.**

I. Introduction

We consider the problem of learning the preferences of an expert agent executing an unknown fixed behavior in an open space. A subject robot is deployed in this environment with the aim of learning the task and aiding the expert agent in performing the task. However, due to a lack of communication protocol between these agents, the learner must understand the expert's behavior from passive observations. We focus on domains with well-defined and repeatable tasks that can benefit from teamwork. Fig. 1 illustrates one such task, which serves as a laboratory abstraction for a robot (e.g., Sawyer) seeking to collaborate with a human in sorting ripe and unripe oranges on a processing line. Collaborative planning requires a joint reward function. Thus, *inverse reinforcement learning* (IRL) [6], [3] becomes particularly relevant as a way to learn the expert's reward function.

Previous work utilizing IRL toward collaboration is sparse, with the vast majority using IRL toward imitating the expert [1]. Natarajan et al. [2] used Ng and Russell's [3] LP-IRL using a policy as input to learn the individual reward functions of multiple agents that are coordinating traffic at intersections. These rewards are combined to perform multiagent optimization in a greedy way. Less closely related is the recent framework for cooperative IRL [4] that formulates a Bayesian Markov game [5] played between a robot seeking to learn the reward function of a human whom may actively teach it. While this framework merges IRL and planning into a single game, our method keeps the two phases separate and shows how a collaborative planning model that uses the output of IRL may be formed automatically.

[1] Maulesh Trivedi was with the Computer Science Department, University of Georgia, Athens, GA 30602 maulesh99@gmail.com

[2] Prashant Doshi is with the Computer Science Department, University of Georgia, Athens, GA 30602 pdoshi@cs.uga.edu

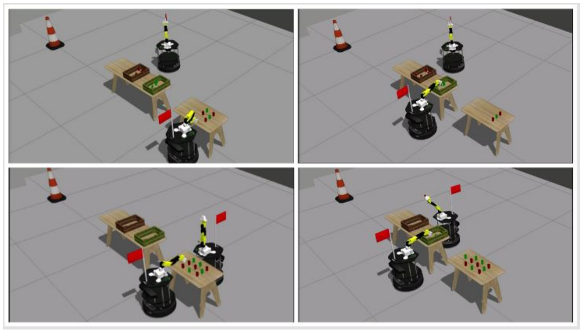
Fig. 1. Subject robot (top) observes another performing the task of sorting colored balls into two bins. The subject robot seeks to team up with the other robot to complete the sorting task faster in the next cycle. Both robots are TurtleBots equipped with Phantom X arms that can grip balls.

Behavior of the expert performing the task is modeled as the solution of a Markov decision process (MDP) with known state and action spaces, and its dynamics but an unknown reward function. The learned preferences are amenable for transfer to the subject robot (with minor adjustments) – as Russell [6] strongly argues – because both robot and observed agent will operate in the same environment, and perform the same task. On acquiring the expert's estimated planning model, this paper shows how it may be used to formulate a joint model for collaborative planning, which would prescribe behavior for both agents allowing spontaneous teamwork.

An early demonstration of impromptu teamwork was in RoboCup where a team that is missing a player could nevertheless play competitively by adding an ad-hoc player [7]. The topic was better formalized by Stone et al. [8] and guidelines for evaluating such teamwork were provided. In particular, the ad hoc agent's goal is to improve the team's performance by spontaneously adapting to or leading the existing agents without being able to modify their behaviors [9], [10]. This paper innovatively positions IRL as a method to learn partial models of these agents from observations. Previous techniques for ad hoc teaming used game theory or partially observable MDPs [11], [12], in which the reward outcomes of joint actions of all agents are assumed to be known or predictable. This requires *full* knowledge of the task being performed [13]. In contexts where the ad hoc agent should exhibit the same but unknown goals and preferences as the existing agents, inverse learning can be used in understanding the collaboration.

We adopt a decentralized MDP [14] as the collaborative model, whose solution is a vector of policies one for each agent. Consequently, the challenge is to find ways to automatically formulate the joint transition and reward models that not only reward joint actions contributing toward task completion, but also manage any interactions between the agents. We evaluate our method on a simulated colored-ball sorting task involving two robots, rendered in Fig. 1, which serves as a stand-in for the industrial application of sorting fruit or vegetables on a postharvest processing line. A crucial impediment to completely formulating the collaborative planning model is the presence of interactions between robots. For example, a detrimental interaction occurs when both robots reach out to grasp the same object; neither of the two robots may ultimately have the object. Our experiments demonstrate that most of the objects were sorted correctly by the spontaneous team, and the two robots working together as a team cut the sorting time down to less than half. The method presented in this paper assumes full observability of the expert and the agents' states. We sought to develop a sound approach for full observability, which is extensible to partial observation settings in the future.

II. Background

The method presented in this paper spans three specific areas of research related to robotic learning and planning. (i) It utilizes IRL to acquire the observed agent's task-driven preferences; (ii) it builds on a Bayesian framework to acquire hidden aspects of the collaborative planning model; and (iii) it uses decentralized MDPs to represent the collaboration problem. To promote understanding, we briefly review the important concepts related to all three areas.

A. Overview of IRL and MaxEnt

Informally, IRL refers to both the problem and method by which an agent learns preferences of another agent that explain the latter's observed behavior [6]. Usually considered an "expert" in the task that it is performing, the observed agent, say E, is modeled as executing the optimal policy of a standard MDP defined as $\langle S_E, A_E, T_E, R_E\rangle$. The learning agent is assumed to perfectly know the parameters of the MDP except the reward function. Consequently, the learner's task is viewed as finding a reward function under which the expert's observed behavior is optimal.

This problem is ill-posed because for any given behavior there are infinitely-many reward functions which align with the behavior. Abbeel and Ng [15], improving on an early method that used a linear program given E's policy [3], present an algorithm that allows E to provide task demonstrations instead of its policy. The reward function is usually modeled as a linear combination of K binary features (although other reward forms have been explored [16]), ϕ: $S_E \times A_E \rightarrow \{0, 1\}$, each of which maps a state from the set of states S_E and an action from the set of E's actions A_E to either a 0 or 1. Note that non-binary feature functions can always be converted into binary feature functions although there will be more of them. These features are either known to or selected by the learner. They may also be learned from data [17] reducing the need for feature engineering. The expert's reward function is then defined as $R_E(s,a) = \sum_{k=1}^{K} \theta_k \cdot \phi_k(s,a)$, where θ_k are the *weights*. The learner's task is reduced to finding a vector of weights that complete the reward function, and subsequently the MDP such that the demonstrations are optimal.

To find the weights, we obtain feature expectations for the expert's demonstration and compare them to those from possible trajectories [18]. A demonstration $\mathcal{X}$ is provided as one or more *trajectories*, which are a sequence of length-T state-action pairs, $(\langle s,a\rangle_1, \langle s,a\rangle_2, \ldots \langle s,a\rangle_T)$, corresponding to an observation of the expert's behavior across T time steps. Feature expectations of the expert are estimated as averages over observed trajectories, $\hat{\phi}_k = \frac{1}{|\mathcal{X}|}\sum_{x\in\mathcal{X}}\sum_{\langle s,a\rangle\in x}\phi_k(s,a)$, where x is a trajectory in the demonstration, $\mathcal{X}$.

Given a set of reward weights the expert's MDP is completed and solved optimally to produce π^*_E. The difference $\hat{\phi}-\phi^{\pi^*_E}$ provides a gradient with respect to the reward weights for a numerical solver. To resolve the degeneracy of this problem, Abbeel and Ng [15] maximize the margin between the value of the optimal policy and the next best policy. While expected to be valid in some contexts, the max-margin approach introduces a bias into the learned reward function in general. To address this, Ziebart et. al. [18] finds the distribution of *maximum entropy* over the set of all trajectories, denoted as $\mathbb{X}$, constrained to match the observed feature expectations.

$$\begin{array}{l} \max\limits_{Pr(\mathbb{X})\in\Delta} \left(-\sum_{x\in\mathbb{X}} Pr(x)\ log\ Pr(x)\right) \\ \textbf{subject to} \quad \sum_{x\in\mathbb{X}} Pr(x) = 1 \\ \sum_{x\in\mathbb{X}} Pr(x)\ \sum_{\langle s,a\rangle\in x}\phi_k(s,a) = \hat{\phi}_k \qquad \forall k \end{array} \tag{1}$$

Here, Δ is the space of all distributions $Pr(\mathbb{X})$. The benefit is that this distribution makes no further assumptions beyond those which are needed to match its constraints and is maximally noncommittal to any one trajectory. A disadvantage is that it becomes intractable for long trajectories because the set of trajectories grows exponentially with length. Given this, another formulation defines the maximum entropy distribution over policies [19], the size of which is large but fixed.

B. Planning (and Learning) under Model Uncertainty

Bayes-adaptive (BA-) MDP [20] allows optimal decision-theoretic planning in Markovian environments under model uncertainty. More specifically, let the transition function $T(s,a,s')$ be unknown. Then, BA-MDP represents the uncertainty over the transition probability using experience counts, $\psi^a_{ss'}$, which denotes the number of times state s transitions to s' on performing action a. The count vector $\boldsymbol{\psi}$ is defined as the vector of counts of all possible transitions for the state and action spaces in consideration.

The expected value of a transition, $T(s,a,s')$, given the count vector $\boldsymbol{\psi}$ is, $T^{\boldsymbol{\psi}}(s,a,s') = \frac{\psi^a_{ss'}}{\sum\limits_{s''\in S}\psi^a_{ss''}}$. The augmented state space S' of the BA-MDP thus becomes $S' = S \times \mathcal{T}$ where $\mathcal{T} = \{\boldsymbol{\psi} \in \mathbb{N}^{|S|^2|A|} | \sum\limits_{s'\in S}\psi^a_{ss'} \geq 0\ \forall(s,a)\}$. Here, $\mathcal{T}$ represents the space of count vectors.

The key to solving BA-MDP lies in the update function for the count vectors. Consider a robot in some state s with count vector $\boldsymbol{\psi}$, which performs action a causing it to transition to a state s'. The count vector $\boldsymbol{\psi}$ is updated as, $\boldsymbol{\psi'} = \boldsymbol{\psi} + \delta^a_{ss'}$, where $\delta^a_{ss'}$ is a vector full of zeros, except for a 1 for the transition count $\psi^a_{ss'}$. Let T' be the transition probability over the augmented space as,

$$T'((s,\psi),a,(s',\psi')) = \begin{cases} T^{\psi}(s,a,s'), & \textit{if } \psi' = \psi + \delta^a_{ss'} \\ 0, & \textit{otherwise} \end{cases} \quad (2)$$

The uncertainty over the parameters is represented by the mixture of Dirichlet distributions, whose expected values are the count vectors. The reward function of the BA-MDP is simply defined as $R'(s,\psi,a) = R(s,a)$. Hence, we can now define the BA-MDP as a tuple of $\langle S', A, T', R, \gamma \rangle$, where the state and the transition functions are augmented with the count vectors to model parameter uncertainty.

III. IRL for Collaborative Planning

The overall process is to employ IRL to learn the goals and preferences of the observed agent. The learned preferences are amenable for transfer to the subject robot (with minor adjustments) – as Russell [6] strongly argues – because both robot and agent will operate in the same environment. The dynamics and transferred preferences are generalized to a collaborative setting, and plugged into a planning framework, which is capable of reasoning about interactions, if any. The solution will be a plan that guides the autonomous robot into a spontaneous and effective collaboration with the agent. We presume that the subject robot possesses the requisite capabilities to observe the agent and perform the task. This presumption is easily satisfied as it involves knowing the capabilities required to perform the task and selecting a robotic platform that possesses them; this is routine for automated production lines. Highlighted activities in Fig. 2 form the foci of steps presented in this section.

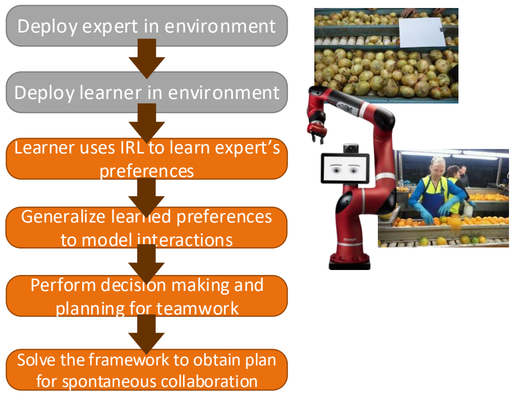
Deploy expert in environment
Deploy learner in environment
Learner uses IRL to learn expert's preferences
Generalize learned preferences to model interactions
Perform decision making and planning for teamwork
Solve the framework to obtain plan for spontaneous collaboration

Fig. 2. The flow of activities involved in the process that would enable an autonomous robot such as Sawyer to spontaneously collaborate with the observed expert agent performing a task such as sorting fruit or vegetables.

A. IRL Model

The first problem that we consider is that of learning the behavior of an expert E executing an unknown fixed policy in an open space. A subject robot L is deployed in this environment with the aim of aiding the expert in its task.

We may define the *state* of a single robot in this pick-and-place environment as a tuple, $s : \langle x, y, inHand, armPos, isTblEmpty \rangle$. The components describe the discretized location of the robot $\langle x, y \rangle$ w.r.t. to a global reference frame in the environment along with its arm position w.r.t. to a local reference frame, and the color of a ball in its hand. *isTblEmpty* defines the status of the pick-up table - it is false if there are balls left to be sorted. We define eight high-level actions for each robot engaged in the sorting task. Actions relating to movement include, goto-pickUp-table, goto-target-1, and goto-target-2. The actions relating to the robot's arm include arm extend, arm close, pickUp, and drop. The noOp action is realized when the robot chooses not to perform any action. Each individual robot's transition function models its dynamics with the probability of completing an action as intended set to 0.9 and the remaining probability mass distributed uniformly among the other actions.

Reward functions are modeled, as mentioned earlier in Section II, as a linear combination of binary feature functions. For our application domain, we define $R(s,a)$ as a combination of the following features:

- *RedInRed(s,a)*: true if agent drops red ball in red box
- *GreenInGreen(s,a)*: true if agent drops green ball in green
- *RedInGreen(s,a)*: true if agent drops red ball in green box
- *GreenInRed(s,a)*: true if agent drops green ball in red box
- *ActionPenalty(s,a)*: true whenever an action is performed

The choice of feature functions indicates that the observing robot does not know how to correctly sort the colored ball, which may involve matching the ball and box colors, or not.

The learner observes the expert as shown in Fig. 1(top) until the latter finishes sorting all colored balls once. Its observations are translated into a sequence of state-action pairs, which are then given as input to an IRL method, such as one of the top performing, MaxEnt, that learns the vector of feature weights which best explains the observations.

B. Collaborative Planning under Model Uncertainty

On learning the agent's preferences fairly accurately, is the subject robot ready to start assisting the agent in sorting the objects as a co-worker, in the next cycle? To answer this question, we hypothesize that the observing robot can automatically formulate a cooperative decentralized MDP, which would correctly model the two-agent sorting problem. With the other's learned policy as input to the decentralized MDP, it could compute its best response behavior to the policy. While each robot's individual transition and reward functions are known, we must understand and model any interactions between them to synthesize the joint model.

Modeling the Joint Behavior: We adopt a decentralized MDP (Dec-MDP) [14] to represent the collaborative planning problem between the two robots. Each robot is assumed to perfectly observe its own state. As the joint state is sufficient for planning in this problem, the problem's state is fully and perfectly observed and a Dec-MDP is appropriate.

The state space of this planning problem is $\mathcal{S} = S_E \times S_L$, where S_E represents the state space of the expert robot and S_L represents the state space of the learner. The action space of the model is $\mathcal{A} = A_E \times A_L$, where A_E and A_L are the actions available to the expert and learner respectively. The states and actions in each of the sets are identical and were previously described in Section III-A.

Notice that the collaboration involves the two robots going about their own tasks except for some states where identical actions may lead to detrimental interactions; for example, both robots seeking to pick up the same ball from the table. Consequently, we may divide the state space into those states where one or more actions lead to an interaction between the two robots, denoted as $\mathcal{S}_I$, and the non-empty subset of remaining states is denoted as $\mathcal{S}_{NI}$. The transition function of the joint model is then obtained as,

$$\mathcal{T}(s,a,s') = \begin{cases} Pr_{NI}(s'|s,a) & \textit{if } s \in \mathcal{S}_{NI} \\ Pr_I(s'|s,a) & \textit{if } s \in \mathcal{S}_I \end{cases} \quad (3)$$

where s and a denote the joint state and action, respectively. Next, $Pr_{NI}(s'|s,a)$ can be defined as,

$$Pr_{NI}(s'_E, s'_L|s_E, s_L, a_E, a_L) = Pr(s'_E|s_E, a_E)\ Pr(s'_L|s_L, a_L)$$

for all $a_E \in A_E$, $a_L \in A_L$. In other words, transitions of the two robots from these states are independent of each other.

The transition function for the interaction states, Pr_I, is challenging because it cannot be synthesized directly from the transition functions of the individual robots as we did previously. Instead, let the learner maintain a count vector for all interacting transitions, $\boldsymbol{\psi}$, which is then included in the state space. Consequently, the augmented state space is now, $\mathcal{S} = S_L \times S_E \times \boldsymbol{\psi}$, analogously to the state space in BA-MDPs (see Section II), and the corresponding framework becomes a *BA-Dec-MDP*. The count of an interacting joint transition in the vector is incremented when the robots engage in such a transition. Subsequently, we may obtain the missing transition probabilities as,

$$Pr_I^{\boldsymbol{\psi}}(s'_E, s'_L|s_E, s_L, a_E, a_L) = \frac{\psi^{a_E,a_L}_{s_E s_L, s'_E s'_L}}{\sum\limits_{s''_E \in S_E} \sum\limits_{s''_L \in S_L} \psi^{a_E,a_L}_{s_E s_L, s''_E s''_L}} \quad (4)$$

Subsequently, as the robots collaborate and engage in actions from states that cause interactions, the counts accumulate and the transition model becomes more accurate via learning.

Next, we address the acquisition of the joint reward function from the reward functions of the individual robots. The learner's reward function is similar to the expert's, which is obtained using IRL. With the two robots behaving independently in the non-interacting states, we obtain,

$$\mathcal{R}(s,a) = R(s_E, s_L, a_E, a_L) = R(s_E, a_E) + R(s_L, a_L),$$

for all $s \in S_{NI}$. This summation of the individual robots' rewards is a standard way of motivating teamwork. It allows the BA-Dec-MDP to favor a positive team reinforcement over an individual's success that could harm others.

Interactions in the particular domain of colored-ball sorting by two robots occur when the two robots attempt to pick up the same ball, leading to undesired *collisions*. Therefore, we may assign a flat negative penalty, say -1, to all interaction states for any action. In other words, $\mathcal{R}(s,a) = -1$ for all $s \in \mathcal{S}_I$.

To summarize, the collaborative planning model is a BA-Dec-MDP whose state space $\mathcal{S}$ is a combination of local states of both robots and a count vector for state transitions; the joint transition model $\mathcal{T}$ is a synthesis of the individual transition models for some transitions, *and learned from experiences for transitions that may lead to interactions*. Analogously, a summation of individual reward functions and a fixed penalty comprise the joint reward function $\mathcal{R}$.

Best Response in a BA-Dec-MDP: We may use value iteration to solve general BA-Dec-MDPs, analogously to how BA-MDPs are solved. However, the complexity of solving Dec-MDPs is, in general, far greater than that of solving MDPs [14]. Fortunately, the following observation helps in significantly reducing the solution complexity.

Because the learner seeks to spontaneously collaborate with the expert without disrupting the expert's behavior, we may use the expert's policy as input to the BA-Dec-MDP.

Recall that learning the reward function of the expert via IRL also allows us to compute the expert's policy that best explains the observations. Let this policy be denoted as π_E. Then, the Bellman equation for the BA-Dec-MDP becomes,

$$V'_{\pi_E}(s) = \max_{a_L \in A_L} \mathcal{R}(s, a_L, \pi_E(s)) + \gamma \sum\nolimits_{s \in \mathcal{S}} \mathcal{T}(s, a_L, \pi_E(s), s') \times V_{\pi_E}(s') \quad (5)$$

Recall that each state $s \in \mathcal{S}$ is factorized into $\langle s_L, s_E, \boldsymbol{\psi} \rangle$. The above equation defines the Bellman update for the dynamic programming solution of our BA-Dec-MDP. We maximize over the actions of the learner while fixing the policy of the expert. The iteration step in the value iteration now loops over the entire joint and count-vector augmented state of both the agents.

With the expert's policy given as input, optimization over the expert's actions is no longer needed. *Thus, the best response computation collapses the BA-Dec-MDP into a BA-MDP, which may be solved far more efficiently.* Specifically, it exponentially reduces the number of computations when compared to a standard Dec-MDP exact solution algorithm.

IV. Experiments and Analysis

Our experiments focused on evaluating the accuracy of the learned reward function of the expert via IRL, followed by evaluating the impact of collaborative planning and plan deployment toward completing the colored ball-sorting task.

A. Experiment setup

We used ROS Indigo on two TurtleBot 2.0 robotic platforms to comprehensively test our end-to-end method. We attached a Phantom X manipulator to each of our robots for picking and placing objects. Each robot is equipped with a Microsoft Kinect 360. We use CMVision integrated with ROS Gazebo for the purpose of collecting observations. We performed a total of 100 trials. Each trial began by placing the learner in the same environment as the expert so that the learner can observe and collect the demonstrated trajectories to perform IRL. The expert follows an optimal policy output from a MDP for sorting 12 balls having two colors in their respective matching boxes. Figure 1 shows our setup in ROS Gazebo.

B. Performance evaluation

Phase one of our experiment deals with recovering the underlying reward structure of the expert robot. The learner starts MaxEnt once the expert has sorted all balls seen on the pick-up table in Fig 1. In order to evaluate IRL's performance, it is tempting to directly compare the reward function of the expert (not known in practice) with that learned by our subject agent. However, a differing reward function (up to affine transformations) may still generate the same optimal policy. This ambiguity is compounded by the lack of sufficient trajectory data. As such, we use a different metric to evaluate.

Choi and Kim [21] suggest comparing the value function of the policy obtained by using the learned reward function with that of the true MDP utilized by the expert. If the policy obtained from the reward function learned by the subject robot, denoted as π_E^L, is identical to the optimal policy followed by the expert, π_E, then the difference between the two value functions for the expert's MDP is zero. Thus, the *inverse learning error (ILE)* is defined as: ILE = $||V^*_{\pi_E} - V_{\pi_E^L}||$. Here, $V^*_{\pi_E}$ is the optimal value function of the expert's MDP and $V_{\pi_E^L}$ is the value function obtained by plugging the learned policy in the expert's MDP. We may also compare the expert's true policy with the learned policy to test for divergence. A simple equivalence test allows us

We achieved an average ILE of 1.55 with a standard deviation of 1.59 that yielded an average policy divergence of just 1.71% with a standard deviation of 0.99%. This low divergence can be attributed to the relatively low number of features used in the reward function, and MaxEnt does not need many long trajectories for small feature spaces. We observed that the learned vector of feature weights emphasizes sorting the colored balls in the correct bins.

In **phase two**, we begin testing the collaborative planning first by analyzing the transition function as estimated by the Bayes-adaptive learning. Importantly, we are interested in that part of the transition function that models the interaction portion of the state space. Which states are included in the interaction space? We consider any state where the two robots' arms can collide with each other. Obviously, this occurs when both robots are at the pick-up table and attempting to pick up the same ball, or attempting to extend or close the arm while the other robot is trying to do the opposite. (We avoid collisions while dropping the ball in the boxes.) This damages the arms and generally fails to complete the task.

The efficacy of using count vectors for learning an accurate transition model is specifically evaluated by allowing both robots to engage in 1,000 episodes. The count vector is initialized using a uniform distribution. An episode is a 3-horizon state-action tree. The considered actions are {extend, close, pickup, drop, noOp}. An episode ends when the horizon is 0, or if a transition results in a collision. Each time an episode ends, we randomly sample a new start state and another begins. The corresponding component of the count vector, $\psi^{a_E,a_L}_{ss'}$, is updated each time the robots experience a new transition. The counts accumulate across episodes.

The learned transition model is compared with the true transition probability for the purposes of evaluation. We use the L_∞ norm to find the maximum divergence between the transition probabilities at each stage of the simulation.

$$L_\infty(\boldsymbol{\psi}) = \max \; |Pr_I^{\psi}(s'_E, s'_L|s_E, s_L, a_E, a_L) - Pr_I(s'_E, s'_L|s_E, s_L, a_E, a_L)|$$

where the maximization is over all states and actions. Eq. 4 gives the transition probabilities on the right-hand side.

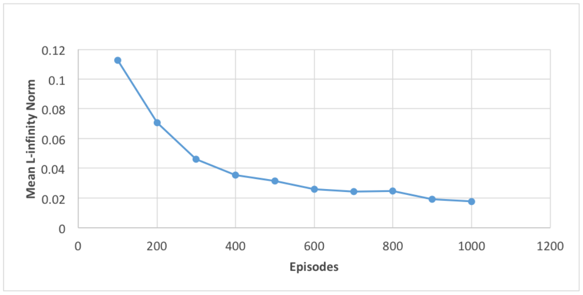

Fig. 3. Count vectors converging to the true transition probabilities for the interaction states.

Figure 3 shows that L_∞ reduces gracefully across the 1,000 episodes for learning the transition model. With more episodes, the worst-case divergence in the learned transition model drops until it stabilizes to a value that is around 0.02. The true probability of colliding when both robots reach out to pick the same object is about 0.95. *A video of a successful run and some collisions is part of the supplementary material.*

Next, the performance of the automatically formulated collaborative planning model is evaluated in two ways. We demonstrate the effectiveness of our method as a function of how many balls are correctly sorted in the two bins and the time taken to do the same. We compare these metrics between a single expert and the two-robot system (expert and learner collaborating as a spontaneous team). A successful test run exhibits quick task completion with a high accuracy in sorting the balls. Each experiment involves sorting all 12 two-colored balls into the separate bins.

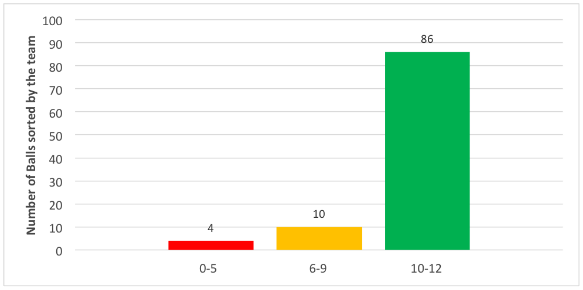

Fig. 4. Performance as measured by the number of balls correctly sorted in each run. The experiment involved a total of 100 runs.

Figure 4 shows the overall accuracy of the team across 100 runs each involving sorting 12 balls. We use a histogram to show the number of runs that correctly sorted an increasing number of the balls. While 86% of the runs sorted almost all the balls (10-12 balls), there were 4 runs that sorted very few balls correctly. A closer look at these runs reveals that in these, peculiar collisions between the arms of the two robots ended with the two arms locked together in a tight grip. Indeed, Fig. 5 shows the various interactions where collisions occured. A deeper analyses revealed that these were either due to sensory noise resulting in an incorrect observation that lead to a wrong action from the policy, or due to a partially incorrect transition model in the BA-Dec-MDP when experience counts are low. A best response of noOp by the learner would have avoided these collisions.

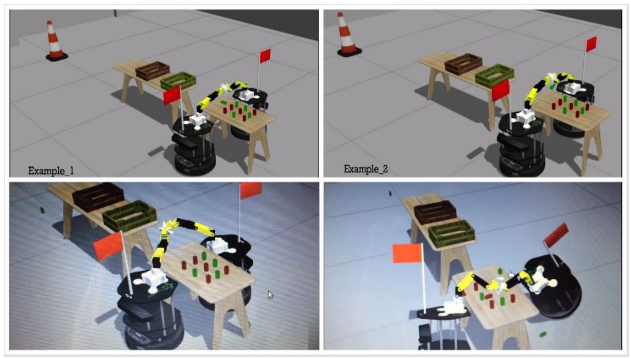

Fig. 5. Various instances of collisions between the learner and expert. The images at the top resulted in minor collisions. The images at the bottom show a perilous situation - both the robots tried to pick up the same ball, and their grippers got locked with each other.

Our ultimate goal is to make the multiagent system more effective than a single agent. A good measure for evaluating this would be to compare the times taken to complete the experiment for a single robot system and a multi-robot team. Figure 6 draws comparisons between the run times for a single and a multi-robot sort. It shows a statistically significant decrease in the amount of time it takes to sort all the balls ($p < 0.05$). An interesting aspect of using ROS Gazebo is the realistic error bars associated with each sort. This volatility is present because the navigation computes a new motion plan each time we send it a goal, and for this, the robot needs to be localized. Fluctuations are observed when the robot fails to generate a new plan as it needs to localize itself again. Interestingly, the two-robot team takes less time and this efficiency accumulates as more balls are sorted. *More specifically, the spontaneous collaboration almost halves the time needed for the expert alone to sort the balls.*

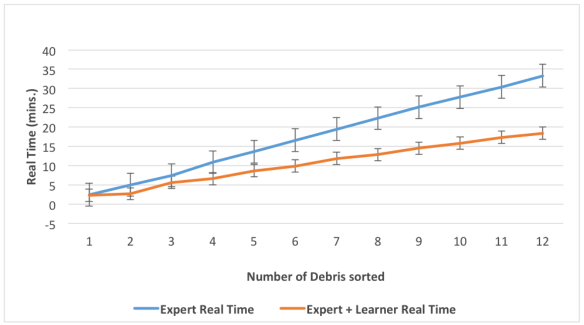

Fig. 6. Time taken in minutes to sort the balls for a single robot vs a multi-robot ad-hoc team.

In summary, our two-robot system achieved a mean reduction of 56.25% in the task completion time. Modeling of interactions in the collaborative planning contributes, in part, to this dramatic improvement. In particular, among the 100 runs shown in Fig. 4, we witnessed 6 major collisions from which no recovery was possible. Another 12 minor collisions were also observed from which the robots recovered and completed the drop. Benefiting from the underlying principled mathematical modeling, we believe that this performance can be replicated to more than two robots in other application domains. Of course, the time taken to solve the IRL and compute the best-response policy may increase concomitantly.

V. Concluding Remarks

Collaboration is a key capability that we require of co-bots. IRL's passive mode of transferring skills is strongly appealing because it significantly mitigates costly human effort in not only manually programming the robot but also in actively teaching the robot through demonstrations. Consequently, this paper is a step toward endowing intelligent co-bots with customizable autonomy to acquire defined and repeatable skills, which align with the observed agent. It is the first in our knowledge to demonstrate end-to-end how robots can transition from being observers to active participants in tasks that benefit from teamwork. Our approach is not tied to any specific IRL technique, and capable methods other than *MaxEnt* should yield similar results. A limitation of our precept of not disturbing the expert's behavior is that the teamwork may suffer from detrimental interactions, which are caused by the expert and cannot be avoided by the learner.

References

[1] B. D. Argall, S. Chernova, M. Veloso, and B. Browning, "A survey of robot learning from demonstration," *Robotics and Autonomous Systems*, vol. 57, no. 5, pp. 469–483, 2009.
[2] S. Natarajan, G. Kunapuli, K. Judah, P. Tadepalli, K. Kersting, and J. Shavlik, "Multi-agent inverse reinforcement learning," in *ICMLA*, 2010, pp. 395–400.
[3] A. Ng and S. Russell, "Algorithms for inverse reinforcement learning," in *ICML*, 2000, pp. 663–670.
[4] D. Hadfield-Menell, S. J. Russell, P. Abbeel, and A. Dragan, "Cooperative inverse reinforcement learning," in *NIPS*, 2016, pp. 3909–3917.
[5] M. Chandrasek, Y. Chen, and P. Doshi, "On Markov games played by Bayesian and boundedly rational players," in *AAAI*, 2017, pp. 437–443.
[6] S. Russell, "Learning agents for uncertain environments (extended abstract)," in *COLT*, 1998, pp. 101–103.
[7] M. H. Bowling and P. McCracken, "Coordination and adaptation in impromptu teams," in *AAAI*, 2005, pp. 53–58.
[8] P. Stone, G. A. Kaminka, S. Kraus, and J. S. Rosenschein, "Ad hoc autonomous agent teams: Collaboration without pre-coordination," in *AAAI*, 2010.
[9] N. Agmon and P. Stone, "Leading ad hoc agents in joint action settings with multiple teammates," in *AAMAS*, 2012, pp. 341–348.
[10] S. Barrett, P. Stone, S. Kraus, and A. Rosenfeld, "Teamwork with limited knowledge of teammates," in *AAAI*, 2013, pp. 102–108.
[11] S. V. Albrecht and S. Ramamoorthy, "A game-theoretic model and best response learning method for ad hoc coordination in multiagent systems," in *AAMAS*, 2013, pp. 1155–1156.
[12] F. S. Melo and A. Sardinha, "Ad hoc teamwork by learning teammates' task," *Journal of AAMAS*, vol. 30, no. 2, pp. 175–219, 2016.
[13] P. Stone, G. A. Kaminka, and J. S. Rosenschein, "Leading a best-response teammate in an ad hoc team," in *Agent-Mediated Electronic Commerce*. Springer, 2010, pp. 132–146.
[14] D. S. Bernstein, R. Givan, N. Immerman, and S. Zilberstein, "The complexity of decentralized control of Markov decision processes," *Math. of Operations Res.*, vol. 27, no. 4, pp. 819–840, 2002.
[15] P. Abbeel and A. Ng, "Apprenticeship learning via inverse reinforcement learning," in *ICML*, 2004.
[16] S. Levine, Z. Popovic, and V. Koltun, "Nonlinear Inverse Reinforcement Learning with Gaussian Processes," in *NIPS*, 2011, pp. 19–27.
[17] S. Levine, Z. Popović, and V. Koltun, "Feature construction for inverse reinforcement learning," in *NIPS*, 2010, pp. 1342–1350.
[18] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey, "Maximum entropy inverse reinforcement learning," in *AAAI*, 2008, pp. 1433–1438.
[19] A. Boularias, O. Kromer, and J. Peters, "Structured apprenticeship learning," *ECML PKDD*, vol. 7524, pp. 227–242, 2012.
[20] M. O. Duff, "Optimal learning: Computational procedures for Bayes-adaptive MDPs," U. of Massachusetts Amherst," PhD Thesis, 2002.
[21] J. Choi and K.-e. Kim, "Inverse Reinforcement Learning in Partially Observable Environments," in *IJCAI*, 2009, pp. 1028–1033.