Proactive Robotic Assistance via Theory of Mind (Technical Appendix)

In this technical appendix, we provide technical background for readers coming from various disciplines, as well as additional details on our approach and experiments. To provide structure for this document, we enumerate the different appendices below:

- **Appendix I:** we provide details on the KD45$_n$ logic.
- **Appendix II:** we provide background on PEKBs and RMLs.
- **Appendix III:** we provide details on RP-MEP's progression operator.
- **Appendix IV:** we specify the modified goal given to the classical planner in the discrepancy resolution algorithm.
- **Appendix V:** we provide additional details on the implementation of our approach within a robotic system.
- **Appendix VI:** we provide a detailed account of our kitchen example from Section III and its encoding in PDKBDDL.
- **Appendix VII:** we provide a PDDL encoding of the kitchen domain that inspired the encoding of our example.
- **Appendix VIII:** we present detailed results from our study.
- **Appendix IX:** we describe the various domains used in our evaluation (Section VI) of the robot's helpfulness.
- **Appendix X:** we discuss additional results from the helpfulness evaluation (Section VI) and the runtime of Algorithm 1.
- **Appendix XI:** we present a comprehensive survey of extant related work.

APPENDIX I
THE MULTI-AGENT MODAL LOGIC KD45$_n$

We discuss the multi-agent modal logic KD45$_n$ [21] which we appeal to in this work. Let Ag and $\mathscr{P}$ be finite sets of agents and atoms, respectively. ϕ and ψ are used to represent formulae. $\top$ and $\bot$ represent *true* and *false*, respectively. The language $\mathscr{L}$ of multi-agent modal logic is generated by the following BNF:

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi' \mid B_i\phi$$

where $p \in \mathscr{P}$, $i \in Ag$, $\phi \in \mathscr{L}$ and $B_i\phi$ means that *"agent i believes ϕ."* The semantics for formulae in $\mathscr{L}$ is given by Kripke models [21] which are triplets, $M = \langle W, R, V\rangle$, containing a set of worlds, accessibility relations between the worlds for each of the agents ($R = \{R_i \mid i \in Ag\}$), and a valuation map, $V : W \to 2^P$. The set of worlds an agent i (at world $w \in W$) considers possible is given by M and the accessibility relations in R_i pertaining to w. R_i is a binary relation on W and is a subset of $W \times W$. A formula ϕ is true in a world w of a Kripke model $M = \langle W, R, V\rangle$, written $M, w \models \phi$, under these conditions:

$M, w \models p$ for an atom p, *iff* $p \in V(w)$,
$M, w \models \neg\phi$, *iff* $M, w \not\models \phi$,
$M, w \models \phi \wedge \psi$, *iff* both $M, w \models \phi$ and $M, w \models \psi$,
$M, w \models B_i\phi$, *iff* $M, w' \models \phi \quad \forall w' \in W$ s.t. $R_i(w, w')$

ϕ is satisfiable if there is a Kripke model M and a world w of M s.t. $M, w \models \phi$. ϕ is said to entail ψ, written $\phi \models \psi$, if for any Kripke model M, $M, w \models \phi$ entails $M, w \models \psi$.

The KD45$_n$ logic is characterized by a particular set of properties of belief and assumes a number of constraints on Kripke models to achieve this [21]. In particular, Kripke models in the KD45$_n$ logic are

Serial - $\forall w \exists v R(w, v)$
Transitive - $R(w, v) \wedge R(v, u) \Rightarrow R(w, u)$
Euclidean - $R(w, v) \wedge R(w, u) \Rightarrow R(v, u)$

with the resulting properties of belief:

$B_i\phi \wedge B_i(\phi \Rightarrow \psi) \Rightarrow B_i\psi$ (K - Distribution)
$B_i\phi \Rightarrow \neg B_i\neg\phi$ (D - Consistency)
$B_i\phi \Rightarrow B_iB_i\phi$ (4 - Positive Introspection)
$\neg B_i\phi \Rightarrow B_i\neg B_i\phi$ (5 - Negative Introspection)

Importantly, the KD45$_n$ system is defined by these properties of belief. In contrast, systems such as S5 logic involve a different set of axioms and therefore cannot facilitate reasoning about agents' beliefs, only their knowledge (i.e., justified true belief). Consequently, S5 logic cannot reason over agents' false beliefs. Previous work that has augmented robots with ToM reasoning and utilized epistemic planning [25] has leveraged an epistemic planning that appeals to S5 and could therefore only reason about agents' knowledge. By appealing to the epistemic planner RP-MEP [34] that enforces the KD45$_n$ axioms, our approach to proactive robotic assistance supports reasoning about agents' (possibly incorrect) beliefs.

APPENDIX II
PROPER EPISTEMIC KNOWLEDGE BASES (PEKBS)

A proper epistemic knowledge base (PEKB) is a set of restricted formulae, called restricted modal literals (RMLs) [37]. An RML is obtained from the following grammar:

$$\phi ::= p \mid B_i\phi \mid \neg\phi$$

where $p \in \mathscr{P}$ and $i \in Ag$. $\mathscr{P}$ and Ag are sets of atoms and agents, respectively, as defined in Section II. The maximum number of nested belief modalities determines the depth of an RML. For instance, $B_i\phi$ has a depth of 1 in addition to the depth of ϕ, where ϕ may hold additional belief modalities. Formally, the depth of an RML is defined [20] as: $depth(p) = 0$ for $p \in P$, $depth(\neg\phi) = depth(\phi)$ and $depth(B_i\phi) = 1 + depth(\phi)$. In addition, a conjunction of RMLs is defined as a set of RMLs, where the set of all RMLs with bounded depth d for a group of agents Ag is denoted as $\mathscr{L}_{RML}^{Ag,d}$. The state of the world is represented by some set of RMLs.

APPENDIX III
RP-MEP PROGRESSION

Definition 5 (Progression (from [34])) *Given a PEKB state ϕ and an action $a = \langle Pre, \{(\gamma_1,\varepsilon_1),...,(\gamma_k,\varepsilon_k)\}\rangle$, $\Gamma = \{(\gamma_1,\varepsilon_1),...,(\gamma_k,\varepsilon_k)\}$, where Pre and each γ_i are PEKBs and each ε_i is an RML, the **progression** of ϕ wrt action a, a PEKB state labelled* $\text{PROG}(a,\phi)$*, is*

$$\begin{aligned}
\text{PROG}(a,\phi) &= (\phi \blacklozenge (R \cup U)) \Diamond Q \\
Q &= \bigcup_{(\gamma_i,\varepsilon_i)\in\Gamma} \{\psi \mid \gamma_i \subseteq \phi \text{ and } \varepsilon_i \models \psi\} \\
R &= \bigcup_{(\gamma_i,\neg\varepsilon_i)\in\Gamma} \{\psi \mid \gamma_i \subseteq \phi \text{ and } \varepsilon_i \models \psi\} \\
U &= \bigcup_{(\gamma_i,\varepsilon_i)\in\Gamma} \{\neg\psi \mid \overline{\gamma_i} \cap \phi = \emptyset \text{ and } \neg\varepsilon_i \models \neg\psi\}
\end{aligned}$$

where $\overline{P}$ is the PEKB that contains the negation of every RML in some PEKB P. $\blacklozenge$ and $\Diamond$ are belief erasure and belief update operators, respectively. Belief update and erasure for PEKBs have been defined and shown to be polynomial time operations[5]*. In case the action a is not executable in ϕ, i.e. Pre $\not\subseteq \phi$,* $\text{PROG}(a,\phi)$ *is undefined. Finally, Q defines the set of literals to be added, R defines the set of literals to be deleted, and U defines the set of uncertain firing literals to be deleted. Uncertain firing occurs when an agent is unsure whether a conditional effect is true and should therefore not believe the effect but must also not believe the opposite.*

For additional details on RP-MEP's progression, we refer readers to [34].

APPENDIX IV
MODIFIED GOAL G' GIVEN TO THE CLASSICAL PLANNER IN THE MODIFIED DISCREPANCY RESOLUTION ALGORITHM

In Section III, we discussed how we modify Shvo et al.'s [38] discrepancy resolution algorithm to handle discpreancy resolution over a set of plans Π, rather than a single plan π as is the case in Shvo et al.'s work. To this end, we modify the epistemic goal corresponding to the discrepancy resolution task:

$$\begin{aligned}
\bigwedge_{\pi_i\in\Pi} \big(& (B_R B_H \text{VALID}(\pi_i, G_H) \wedge B_R \text{VALID}(\pi_i, G_H)) \vee \\
& (B_R B_H \neg\text{VALID}(\pi_i, G_H) \wedge B_R \neg\text{VALID}(\pi_i, G_H)) \big)
\end{aligned}$$

To reflect the modified discrepancy resolution epistemic goal, we modify G', the goal given to the classical planner in Line 6 of Shvo et al.'s Algorithm 1 [38]. In Line 3 of Shvo et al.'s Algorithm 1, the domain, plan and goal are classically encoded by RP-MEP. π_c and G_c are then the classically encoded plan and goal corresponding to the plan and goal π and G given to the discrepancy resolution algorithm . In our modified version of the algorithm, we use an implementation[6] of the regression

[5]Miller, T., & Muise, C. J. (2016, July). Belief Update for Proper Epistemic Knowledge Bases. In IJCAI (pp. 1209-1215).

[6]J. Rintanen, "Regression for classical and nondeterministic planning," in ECAI 2008. IOS Press, 2008, pp. 568–572.

operator REG for classical planning with conditional effects, and compute $\phi_i = \text{VALID}(\pi_{ic}, G_{ic})$ for every plan π_i in Π. The goal G' given to the classical planner is then

$$\bigwedge_{\pi_i \in \Pi} \bigvee_{\phi_d \in \text{DNF}(\phi_i)} \left(\bigwedge_{\phi_{dc} \in \phi_d} \mathscr{C}(B_R B_H \mathscr{D}(\phi_{dc})) \wedge \mathscr{C}(B_R \mathscr{D}(\phi_{dc})) \right) \vee$$
$$\bigvee_{\phi_d \in \text{DNF}(\neg \phi_i)} \left(\bigwedge_{\phi_{dc} \in \phi_d} \mathscr{C}(B_R B_H \mathscr{D}(\phi_{dc})) \wedge \mathscr{C}(B_R \mathscr{D}(\phi_{dc})) \right),$$

where $\mathscr{C}()$ and $\mathscr{D}()$ are mapping functions from RMLs in the domain to propositional fluents in the classically encoded domain (and vice versa). More details are in [38].

Finally, recall our kitchen example discussed in Section III. A valid discrepancy resolving plan would be for Pepper to inform Alice that the bowl is not in Cabinet1 and to move the bowl from Cabinet1 to some other location, such that the assistive solution generated in Line 8 is no longer valid. This is a valid discrepancy resolving plan since following its execution, Pepper will not perceive discrepancies between its beliefs and Alice's beliefs, pertaining to the validity of plans in Π. This solution to the problem is, however, undesirable since Alice in this case will not know how to achieve her goal (that is, she will have to replan and search for the bowl).

Shvo et al. [38] discuss this issue that arises due to the generality of their formulation. As a solution, they propose to constrain plan generation appropriately. In our case, we modify the discrepancy resolution goal G' further to ensure that at least one plan π_i in the set of plans Π is believed by the robot R to be valid, and moreover that the robot believes that the human H believes that π_i is valid. To this end we add the following conjunct to the goal G' given to the classical planner:

$$\bigvee_{\pi_i \in \Pi} \bigvee_{\phi_d \in \text{DNF}(\phi_i)} \left(\bigwedge_{\phi_{dc} \in \phi_d} \mathscr{C}(B_R B_H \mathscr{D}(\phi_{dc})) \wedge \mathscr{C}(B_R \mathscr{D}(\phi_{dc})) \right)$$

That is, we require that at least one plan $\pi_i \in \Pi$ is believed to be valid both by the robot and by the human (from the robot's perspective). Returning to our example, this ensures that Pepper will either (1) inform Alice that the bowl is no longer in Cabinet1 but rather in Cabinet2; or (2) move the bowl from Cabinet2 to Cabinet1. Importantly, the aforementioned undesirable discrepancy resolving plan (involving Pepper making Alice's presumed plan, as well as the assistive solution, invalid) no longer achieves G'.

APPENDIX V
IMPLEMENTATION DETAILS

In this section we present additional details regarding the implementation of our approach within a semi-humanoid robot.

A. Line 3 – Perception Module

To interface the epistemic planner with Pepper's hardware we first cross-compile ROS, Naoqi driver, and their dependencies so that it can be run directly on Pepper's onboard computer. The machine connected to Pepper via LAN subscribes to compressed RGB streams which are processed by the perception module. To run our experiments, we use a 6GB GTX1650 Ti Nvidia GPU and a 9th gen Intel i7 processor.

As discussed in Section IV, we use MonoLoco [49] (a lightweight pre-trained neural network) to detect a person's 3D position and orientation from Pepper's RGB camera. MonoLoco is a deep learning based model that takes as input 2D keypoint detections from OpenPifPaf[7] and outputs 2D bounding-box pixel coordinates, 3D position and orientation estimates along with their uncertainties. The estimated uncertainties can further be used to perform tracking of a particular person. For every frame of RGB data received, the perception module runs MonoLoco to get detections of people. For every instance of people detected in the image, we compute AlignedReID features. These features are compared between all people in the current frame, and also with features extracted from previous frame in order to associate instances of the same people and differentiate between different people.

Event detection: As discussed in Section IV, we map the processed observations (obtained from the various perception algorithms) to agents from the set of agents Ag (e.g., Alice), atoms in $\mathscr{P}$ (e.g., Soup1), and one of 7 possible events: pick up, put down, open (close) cabinet, enter (leave) room, and shift gaze. We then map these to an action in the set of actions $\mathscr{A}$ (e.g., *pickUp*(Alice, Soup1)).

We wrote a simple logic that returns one of these 7 events based on (1) a change in the proximity of a person to objects and locations in the room. Our logic is scenario-specific. For instance, in our 'charger' scenario (see Appendix IX-D), our logic detects that a person is picking up the phone charger if that person is close enough to the charger; (2) whether a person is detected by Pepper's camera or not (and whether that person was previously (not) detected by Pepper's camera); and (3)

[7]S. Kreiss, L. Bertoni, and A. Alahi, "OpenPifPaf: Composite fields for semantic keypoint detection and spatio-temporal association." IEEE, 2021.

a change in the orientation estimation of a person given by MonoLoco. That is, if a person's predicted orientation changes we then estimate a person's gaze direction and send a *shift gaze* action to RP-MEP. This allows Pepper to perform visual perspective taking – the ability to see the world from another person's perspective [50]. See Figure 1 for a sample output of MonoLoco where the person is looking in 3 different directions. See Appendix IX-D for details on how visual perspective taking is implemented in RP-MEP.

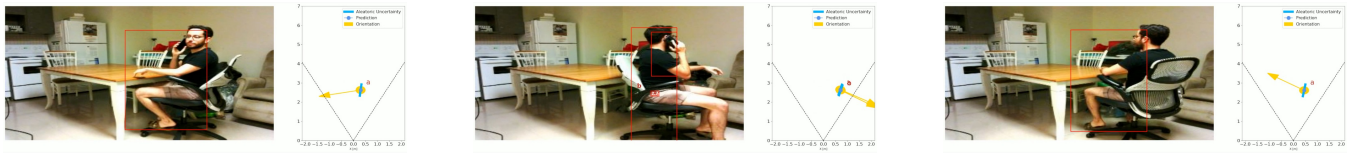

Fig. 1. Sample output from MonoLoco's orientation estimation. In each image, the person is gazing in a different direction and the orientation estimation changes accordingly.

B. Lines 4-10 – ToM Reasoning

1) Line 4 - Progression: In Line 4, the state S is progressed using RP-MEP's plan validation feature that accepts as input a plan, a goal, a domain, and a state, and determines whether the plan achieves the goal, while also returning the final state following the execution of the plan.

2) Line 6 - Plan recognition: In Line 6, Shvo et al.'s epistemic plan recognition algorithm is implemented as described in [45]. In particular, to compute the Δ with RP-MEP in Line 4 of their algorithm, we provide the Fast Downward classical planner [57] with the encoded PDDL files and run the planner, twice for each goal in $\mathscr{G}$, with a satisficing configuration to improve runtime.

3) Line 8 - Generating an assistive solution: In Line 8, RP-MEP is used to solve the MEP problem $\langle\langle \mathscr{P}, \mathscr{A}, Ag\rangle, S, B_R G_H\rangle$ and generate an assistive solution for the human's presumed goal, G_H. Fast Downward is used with an admissible heuristic, to ensure optimal plans are computed.

4) Line 10 - Discrepancy resolution: In Line 10, Shvo et al.'s modified discrepancy resolution algorithm is implemented as described in [38] and in Section III. In Line 3 of the modified algorithm, RP-MEP is given PDKBDDL files and outputs classically encoded PDDL files, and in Line 6 Fast Downward is given the encoded PDDL files and called with an admissible heuristic that supports conditional effects and disjunctive goals, to ensure optimal plans are computed. Like Shvo et al., we also make use of the SymPy[8] Python library to convert regression formulae to DNF and to compute their negation.

C. Line 11 – Plan Execution by Pepper

In this work, Pepper is able to: (1) navigate to different places in the scene by leveraging the map built using RTAB-Map, Pepper's depth sensors, and ROS's navigation stack; (2) interact with objects – since robotic object manipulation is a research problem in and of itself, for the purposes of the video recordings in our study we hardcoded Pepper's limb movement so that it appears as though Pepper is successfully manipulating objects (e.g., closing a door or picking up a phone charger); (3) communicate (e.g., Pepper informing Alice of the bowl's location in our example). To realize this, the content of an action is processed using a number of simple natural language templates. Then, the text is given to Pepper's text-to-speech module and/or displayed on Pepper's tablet.

1) Executing world-altering actions: As discussed in Section IV, since real-world object manipulation is a research problem in and of itself (as well as Pepper having extremely limited dexterity), for the purposes of our work and the video recordings used in our study, we hardcoded Pepper's limb movement so that it appears as though Pepper is successfully manipulating objects.

For instance, in our 'charger' scenario (see Appendix IX-D) Pepper – as part of the execution of the automatically generated discrepancy resolving plan – is meant to pick up a phone charger that is connected to a power outlet. To record the video, we hardcoded Pepper's movement so that it leans closer to the power outlet and lifts its arm. Then, the recording was paused and resumed after we placed the charger in Pepper's hand. Moreover, in the 'corridor' scenario Pepper is meant to close a door in order to resolve the human's discrepancy. Similarly to the charger scenario, we hardcoded Pepper's movement so that it leans close to the door and lifts its arm. Then, we closed the door ourselves and had Pepper return to its neutral body position.

[8] http://sympy.org/

2) Executing communication actions: As discussed in Section IV, Pepper can communicate with other agents (e.g., Pepper informing Alice of the bowl's location in our example). If a discrepancy resolving plan generated in Line 10 of Algorithm 1 contains communication actions then their content is processed using a number of simple natural language templates. For instance, recall the inform action from the discrepancy resolving plan π' in our kitchen example:

$$[\mathit{inform}(\text{Pepper, Alice, } \neg \mathit{in}(\text{Bowl1,Cabinet1}) \wedge \mathit{in}(\text{Bowl1,Cabinet2}))].$$

This action is processed and converted to the English sentence: "If you're looking for Bowl1, it's in Cabinet2, not in Cabinet1". In our study we produced video recordings of Pepper implementing Algorithm 1. However, due to incredibly poor acoustics in the filming location, we recorded our own voice (and distorted it) instead of recording Pepper's text-to-speech module.

APPENDIX VI
ENCODING THE KITCHEN EXAMPLE FROM SECTION III IN PDKBDDL

Here we detail how our kitchen example described in Section III (and used in our evaluation in Section VI) was encoded using the PDKBDDL format used by the epistemic planner RP-MEP. We moreover offer a more concrete view of the workings of Algorithm 1. For the sake of readability, the domain seen here is partial and contains a subset of the actions found in the full domain. Moreover, to better illustrate the PDKBDDL format, some actions are grounded versions of the actions used in our experiments.

As defined in Section II, a MEP problem comprises a domain, an initial state, and a goal condition. Correspondingly, the epistemic planner RP-MEP (as is typically the case in automated planning) accepts two components – a domain file and a problem file, both encoded in PDKBDDL. The former contains definitions of the agents, actions, predicates, and types of objects in the environment. The latter specifies the initial state of the world and the goal condition.

domain-kitchen.pdkbddl

```
(define (domain kitchen)

    ; This specifies the agents in the set of agents Ag
    (:agents alice bob)

    (:types room thing cabinet)

    (:predicates
       (at ?ag - agent ?r - room)
       (in ?obj - thing ?cab - cabinet)
       (holding ?ag - agent ?obj - thing)
       (atRobot ?r - room)
    )


    (:action takeObjOutOfCabinet
      :derive-condition   (at $agent$ kitchen)
      :parameters         (?ag - agent ?obj - thing ?cab - cabinet)
      :precondition       (and (in ?obj ?cab) (open ?cab))
      :effect             (and

                                (not (in ?obj ?cab))

                                (holding ?ag ?obj)

                           )
  )

  (:action informObjLocation
      :derive-condition   always
      :parameters         (?a1 - agent ?obj - thing ?cab - cabinet)
      :precondition       (and (in ?obj ?cab))
      :effect             (and
```

```
                                [?a1](in ?obj ?cab))

    )

    (:action informObjNotInLocation
      :derive-condition   always
      :parameters         (?a1 - agent ?obj - thing ?cab - cabinet)
      :precondition       (and (not (in ?obj ?cab)))
      :effect             (and

                                [?a1](!in ?obj ?cab))

    )


)
```

Since planning in RP-MEP is from the perspective of a single agent (called the *root agent* by Muise et al. [20], [34]), all fluents are implicitly preceded by the beliefs of that agent. In our kitchen example, Pepper resolves all discrepancies and so planning in RP-MEP is done from its perspective. For instance, in the *informObjLocation* action, the effect [?a1](in ?obj ?cab) is implicitly preceded by [Pepper] such that the effect of the action is that Pepper believes that agent a1 believes that the object *obj* is in cabinet *cab*.

The other component of a MEP problem specified in PDKBDDL and given to RP-MEP is the problem file. The problem file specifies the different objects, the initial state of the world, and the goal.

problem-kitchen.pdkbddl

```
{include:domain-kitchen.pdkbddl}


(define (problem prob1)
  (:domain kitchen)
    (:objects kitchen - room
        soup1 bowl1 - thing
        cabinet1 cabinet2 cabinet3 - cabinet
                            )
    (:projection )
    (:depth 1)
    (:task valid_generation)
    (:init-type complete)

  (:init
  (!at alice kitchen)
  (at bob kitchen)
  (in soup1 cabinet3)
  (in bowl1 cabinet1)
  [alice](in bowl1 cabinet1)
  [bob](in bowl1 cabinet1)
  [alice](in soup1 cabinet3)
  [bob](in soup1 cabinet3)
  .
  .
  .
  .
)
```

Recall that in our example, Pepper (via the perception module) observed Alice place a bowl (Bowl1) in a certain cabinet (Cabinet1) and leave the room. The problem file shown above specifies the state of the world after Alice's actions. We can

see that the state of the world (from Pepper's perspective) is such that everyone believes that the bowl is in Cabinet1 and the can of soup (Soup1) is in Cabinet3. Moreover, Pepper believes that Alice is not in the kitchen and that Bob is in the kitchen.

After Bob is observed opening Cabinet1, taking Bowl1, placing Bowl1 in Cabinet2, and leaving the kitchen, we call RP-MEP's plan validation feature that accepts as input a plan, a goal, a domain, and a state, and determines whether the plan achieves the goal, while also returning the final state following the execution of the plan. The plan given to the validation feature comprises Bob's observed actions. The resulting state returned by the validation feature is the following:

problem-kitchen.pdkbddl

```
 .
 .
 .
  (:init
  (!at alice kitchen)
  (!at bob kitchen)
  (in soup1 cabinet3)
  (in bowl1 cabinet2)
  [alice](in bowl1 cabinet1)
  [bob](in bowl1 cabinet2)
  [alice](in soup1 cabinet3)
  [bob](in soup1 cabinet3)
  .
  .
  .
  .
)
```

As discussed in Section III, after observing Bob's actions, Pepper believes that Alice holds a false belief pertaining to the bowl's location – whereas Pepper believes that Alice believes it is still in Cabinet1, Pepper believes that it is in Cabinet2. As discussed in Section III, Pepper's reasoning is done automatically by RP-MEP using the conditioned mutual awareness mechanism [34].

Going back to domain-kitchen.pdkbddl, note the value of *derive-condition* in the *takeObjOutOfCabinet* action – (at $agent$ kitchen). (at $agent$ kitchen) specifies the condition for *mutual awareness*. In this case, agents believe the effects of the *takeObjOutOfCabinet* action if they are in the kitchen, where this *grounded*[9] action is performed. As discussed in Section III, conditioned mutual awareness can handle both first- as well as higher-order ToM reasoning. This depends on the depth of nested belief that RP-MEP is configured to reason with (the depth of nested belief is specified in the PDKBDDL problem file (discussed below)). In the kitchen example, if the specified depth of nested belief in the problem file is 1, then if Alice and Bob are in the kitchen and Bob picks up Bowl1 from Cabinet1, then Alice will believe the bowl is no longer in Cabinet1. However, in our example Alice is not in the kitchen and therefore will still believe that the bowl is in Cabinet1 even after Bob takes the bowl out of Cabinet1. Since all reasoning is done from Pepper's perspective and since Pepper believes that Alice was not in the room when Bob moved the bowl, Pepper will believe after observing Bob's action that Alice's beliefs about the bowl's location did not change.

In Line 6 our algorithm performs plan recognition. As discussed in Section III, the partial set of possible goals the human H may be pursuing is $\mathscr{G} = \{made\_soup, made\_coffee, \dots\}$. Our domain and set of possible goals are inspired by the popular Kitchen domain[10], often used as a goal recognition benchmark [43]. See the full encoding of the classical planning domain in PDDL in Appendix VII. $\mathscr{G}$ in our case includes the 'soup making' goal (which is achieved if the agent picks up a bowl and a can of soup) and a 'coffee making' goal (which is achieved if the agent picks up coffee, creamer, sugar, and a mug). After observing Alice returning to the kitchen and taking the can of soup from Cabinet3, Shvo et al.'s [45] plan recognition algorithm (using RP-MEP) returns the plan π_H:

Alice's presumed plan

```
1. openCabinet_alice_cabinet1
2. takeObjOutOfCabinet_alice_bowl1_cabinet1
```

The plan recognition algorithm also returns the human's presumed goal $G_H = made\_soup$. This is because the sequence of observations O includes *enterRoom*(Alice), *open*(Alice, Cabinet3), and *pickUp*(Alice, Soup1), and Shvo et al.'s [45]

[9]For the lifted action, the derive-condition would be (at $agent$?r), where r is the room where the action is performed.

[10]Wu, J., Osuntogun, A., Choudhury, T., Philipose, M., & Rehg, J. M. (2007, October). A scalable approach to activity recognition based on object use. In 2007 IEEE 11th international conference on computer vision (pp. 1-8). IEEE.

algorithm forces the planner to generate plans that satisfy O. That is, achieving *made_soup* while satisfying the sequence of observations is 'cheaper' than achieving *made_coffee* while satisfying the observations (since a plan that satisfies O and achieves *made_coffee* would include the action of picking up the can of soup that is redundant since it does not contribute towards the optimal achievement of *made_coffee*).

As discussed in Section III, π_H conforms with the sequence of observations and achieves the goal from the perspective of the observed agent, Alice in our case. Since Alice holds a false belief about the location of Bowl1, her plan to make soup involves obtaining the bowl from Cabinet1 (rather than Cabinet2). Her plan will of course fail because of her false belief.

To obtain an assistive solution in Line 8, RP-MEP is tasked with solving the MEP problem $\langle\langle \mathscr{P}, \mathscr{A}, Ag\rangle, S, B_R G_H\rangle$ by generating a plan π_{asst} that comprises only actions performed by the human H. Importantly, Pepper believes that π_{asst} achieves the human's goal *made_soup*. The goal in the PDKBDDL problem file is

```
made_soup
```

Recall that *made_soup* is implicitly preceded by [Pepper]. The assistive solution π_{asst} generated by RP-MEP is then

Assisstive solution for Alice's presumed goal

```
1. openCabinet_alice_cabinet2
2. takeObjOutOfCabinet_alice_bowl1_cabinet2
```

Since Pepper believes that Bowl1 is in Cabinet2, π_{asst} involves Alice taking the bowl from Cabinet2, rather than Cabinet1.

Finally, in Line 10 Π is populated with π_H and π_{asst} and given (along with G_H) to Shvo et al.'s modified discrepancy resolution algorithm [38]. To generate the communicative discrepancy resolving plan π' shown in Section III, we allow Pepper to only communicate with other agents, without altering the environment. Shvo et al.'s modified discrepancy resolution algorithm, using RP-MEP, therefore generates the following discrepancy resolving plan

Discrepancy resolving plan

```
1. informObjNotInLocation_alice_bowl1_cabinet1
2. informObjInLocation_alice_bowl1_cabinet2
```

That is, the discrepancy resolving plan involves Pepper informing Alice about the location of Bowl1 which resolves both discrepancies perceived by Pepper between its beliefs and Alice's beliefs – $\textsc{valid}(\pi_H, G_H)$ and $\textsc{valid}(\pi_{asst}, G_H)$.

Appendix VII
Classical planning Kitchen domain

We include the full PDDL encoding of the kitchen domain that inspired the encoding of our kitchen domain and set of possible goals.

Kitchen domain

```
(define (domain kitchen)
        (:requirements :strips :typing :action-costs)
        (:types object useable container )
        (:constants
                water_jug kettle cloth tea_bag bowl milk
                cereal creamer cup sugar coffee cheese plate
                bread butter knife peanut_butter spoon
                 juice  dressing salad_tosser lunch_bag - object
                 toaster  - useable
                foodcupboard equipmentcupboard fridge draw - container)
        (:predicates
                (taken ?o - object)
                (used ?o - useable)
                (in ?o - object ?c - container)
                (is_open ?c - container)
                (water_boiled)
                (made_tea)
                (made_cereals)
                (made_coffee)
                (made_cheese_sandwich)
                (made_toast)
                (made_buttered_toast)
```

```
        (made_peanut_butter_sandwich)
        (lunch_packed)
        (made_breakfast)
        (made_salad)
        (made_dinner)
        (dummy)
)
(:functions
        (total-cost) - number
)
(:action OPEN
        :parameters (?c - container )
        :precondition (and
                        (not (is_open ?c) )
                )
        :effect ( and
                        (is_open ?c)
                        (increase (total-cost) 7)
                )
)
(:action CLOSE
        :parameters (?c - container )
        :precondition (and
                                (is_open ?c)
                        )
        :effect ( and
                        (not (is_open ?c))
                        (increase (total-cost) 1)
                )
)
(:action TAKE
        :parameters (?obj - object )
        :precondition (and (dummy)
                                                (not (exists (?c - container)
                                        (and (in ?obj ?c))
                                        ))
                                )
        :effect ( and
                        (taken ?obj)
                        (increase (total-cost) 1)
                )
)
(:action TAKE
        :parameters (?obj - object ?c - container)
        :precondition (and (dummy)
                                            (in ?obj ?c)
                                            (is_open ?c)
                                            )
        :effect ( and
                        (taken ?obj)
                        (not (in ?obj ?c))
                        (increase (total-cost) 1)
                )
)
(:action USE
        :parameters (?obj - useable )
        :precondition (and (dummy)
```

```
                                    )
        :effect (and
                        (used ?obj)
                        (increase (total-cost) 1)
                )
)
(:action ACTIVITY-Boil-Water
        :parameters ()
        :precondition         (and
                                (taken water_jug)
                                (taken kettle)
                                (taken cloth)
                        )
        :effect                (and
                                (water_boiled)
                                (increase (total-cost) 1)
                        )
)
(:action ACTIVITY-Make-Tea
        :parameters ( )
        :precondition        (and
                                (taken tea_bag)
                                (taken cup)
                                (taken sugar)
                                (water_boiled)
                        )
        :effect                (and
                                (made_tea)
                                (increase (total-cost) 1)
                        )
)
(:action ACTIVITY-Make-Tea
        :parameters ( )
        :precondition        (and
                                (taken tea_bag)
                                (taken cup)
                                (taken sugar)
                                (taken milk)
                                (water_boiled)
                        )
        :effect                (and
                                (made_tea)
                                (increase (total-cost) 1)
                        )
)
(:action ACTIVITY-Make-Tea
        :parameters ( )
        :precondition        (and
                                (taken tea_bag)
                                (taken cup)
                                (water_boiled)
                        )
        :effect                (and
                                (made_tea)
                                (increase (total-cost) 1)
                        )
)
```

```
(:action ACTIVITY-Make-Cereals
        :parameters ()
        :precondition        (and
                                (taken bowl)
                                (taken cereal)
                                (taken milk)
                        )
        :effect                (and
                                (made_cereals)
                                (increase (total-cost) 1)
                        )
)
(:action ACTIVITY-Make-Coffee
        :parameters ()
        :precondition        (and
                                (taken cup)
                                (taken coffee)
                                (taken creamer)
                                (taken sugar)
                                (water_boiled)
                        )
        :effect                (and
                                (made_coffee)
                                (increase (total-cost) 1)
                        )
)
(:action ACTIVITY-Make-Coffee
        :parameters ()
        :precondition        (and
                                (taken cup)
                                (taken coffee)
                                (taken milk)
                                (taken sugar)
                                (water_boiled)
                        )
        :effect                (and
                                (made_coffee)
                                (increase (total-cost) 1)
                        )
)
(:action ACTIVITY-Make-Cheese-Sandwich
        :parameters ()
        :precondition        (and
                                (taken bread)
                                (taken cheese)
                                (taken plate)
                        )
        :effect                (and
                                (made_cheese_sandwich)
                                (increase (total-cost) 1)
                        )
)
(:action ACTIVITY-Make-Toast
        :parameters ( )
        :precondition        (and
                                (taken bread)
                                (used toaster)
```

```
                        )
        :effect                 (and
                                 (made_toast)
                                 (increase (total-cost) 1)
                        )
)
(:action ACTIVITY-Make-Buttered-Toast
        :parameters ( )
        :precondition        (and
                                 (made_toast)
                                 (taken butter)
                                 (taken knife)
                        )
        :effect                 (and
                                 (made_buttered_toast)
                                 (increase (total-cost) 1)
                        )
)
(:action ACTIVITY-Make-Peanut-Butter-Sandwich
        :parameters ()
        :precondition        (and
                                 (taken bread)
                                 (taken peanut_butter)
                                 (taken knife)
                                 (taken plate)
                        )
        :effect                 (and
                                 (made_peanut_butter_sandwich)
                                 (increase (total-cost) 1)
                        )
)
(:action ACTIVITY-Pack-Lunch
        :parameters ( )
        :precondition        (and
                                 (taken lunch_bag)
                                 (made_cheese_sandwich)
                        )
        :effect                 (and
                                 (lunch_packed)
                                 (increase (total-cost) 1)
                        )
)
(:action ACTIVITY-Pack-Lunch
        :parameters ()
        :precondition        (and
                                 (taken lunch_bag)
                                 (made_peanut_butter_sandwich)
                        )
        :effect                 (and
                                 (lunch_packed)
                                 (increase (total-cost) 1)
                        )
)
(:action ACTIVITY-Make-Breakfast
        :parameters ( )
        :precondition        (and
                                 (made_tea)
```

```
                                    (taken spoon)
                                    (made_cereals)
                                    (made_buttered_toast)
                            )
                :effect                 (and
                                    (made_breakfast)
                                    (increase (total-cost) 1)
                            )
        )
        (:action ACTIVITY-Make-Breakfast
                :parameters ( )
                :precondition           (and
                                    (made_coffee)
                                    (taken spoon)
                                    (made_cereals)
                                    (made_buttered_toast)
                            )
                :effect                 (and
                                    (made_breakfast)
                                    (increase (total-cost) 1)
                            )
        )
        (:action ACTIVITY-Make-Salad
                :parameters ( )
                :precondition           (and
                                    (taken bowl)
                                    (taken plate)
                                    (taken dressing)
                                    (taken salad_tosser)
                            )
                :effect                 (and
                                    (made_salad)
                                    (increase (total-cost) 1)
                            )
        )
        (:action ACTIVITY-Make-Salad
                :parameters ( )
                :precondition           (and
                                    (taken bowl)
                                    (taken plate)
                                    (taken salad_tosser)
                            )
                :effect                 (and
                                    (made_salad)
                                    (increase (total-cost) 1)
                            )
        )
        (:action ACTIVITY-Make-Dinner
                :parameters ()
                :precondition           (and
                                    (made_salad)
                            )
                :effect                 (and
                                    (made_dinner)
                                    (increase (total-cost) 1)
                            )
        )
```

```
        (:action ACTIVITY-Make-Dinner
                :parameters ( )
                :precondition         (and
                                        (made_cheese_sandwich)
                                )
                :effect                 (and
                                        (made_dinner)
                                        (increase (total-cost) 1)
                                )
        )
        (:action ACTIVITY-Make-Dinner
                :parameters ()
                :precondition         (and
                                        (made_salad)
                                        (made_cheese_sandwich)
                                )
                :effect                 (and
                                        (made_dinner)
                                        (increase (total-cost) 1)
                                )
        )

)
```

APPENDIX VIII
USER STUDY - RESULTS

In this section we present detailed results from our study. We separate the section into 1 subsection for internal consistency results and 3 additional subsections, one for each scenario used in our study – charger, kitchen, and corridor. For more details on these domains, see Appendix IX-D. For each scenario, we report the results for the different PSI scales used in the study – RC, PC, AC, and HLP.

A. *Internal Consistency*

The PSI scales we used in the study were tested for reliability. All scales consisted of four questions [58]. All scales had internal consistency, as determined by Cronbach's alpha: AC (α = 0.744), RC (α = 0.919), PC (α = 0.946) and HLP (α = 0.945).

B. *Charger scenario*

1) Recognizes Human Cognitions (RC): Shapiro-Wilk tests showed a significance departure from the normality ($p < .05$) for both conditions. Therefore, we conducted a Mann-Whitney U test to determine if the perception of a robot being able to recognize the cognition of humans was different depending on condition. The differences between conditions were statistically significant with the **ToM** condition (mean rank = 51.45) being higher than the **ToM-Def** condition (mean rank = 30.09), U = 372.500, $p < .001$.

2) Predicts Human Cognitions (PC): Shapiro-Wilk tests showed a significance departure from the normality ($p < .05$) for both conditions. Therefore, we conducted a Mann-Whitney U test to determine if the perception of a robot being able to predict the cognition of humans was different depending on condition. The differences between conditions were statistically significant with the **ToM** condition (mean rank = 49.81) being higher than the **ToM-Def** condition (mean rank = 31.65), U = 436.500, $p < .001$.

3) Adapts to Human Cognitions (AC): Shapiro-Wilk tests showed a significance departure from the normality ($p < .05$) for both conditions. Therefore, we conducted a Mann-Whitney U test to determine if the perception of a robot being able to adapt to the cognition of humans was different depending on condition. The differences between conditions were statistically significant with the **ToM** condition (mean rank = 54.00) being higher than the **ToM-Def** condition (mean rank = 27.66), U = 273.000, $p < .001$.

4) Helpful (HLP): Shapiro-Wilk tests showed a significance departure from the normality ($p < .05$) for both conditions. Therefore, we conducted a Mann-Whitney U test to determine if the perception of a robot being helpful was different depending on condition. The differences between conditions were statistically significant with the **ToM** condition (mean rank = 52.68) being higher than the **ToM-Def** condition (mean rank = 28.91), U = 324.500, $p < .001$.

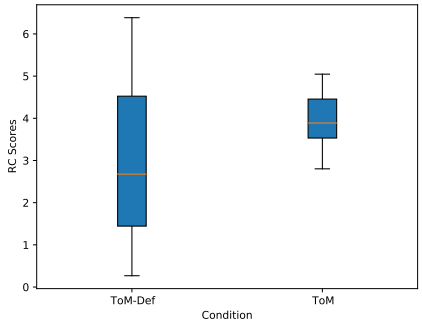

Fig. 2. Charger scenario - RC scores in the **ToM** condition were significantly higher than the **ToM-Def** condition ($p < .001$).

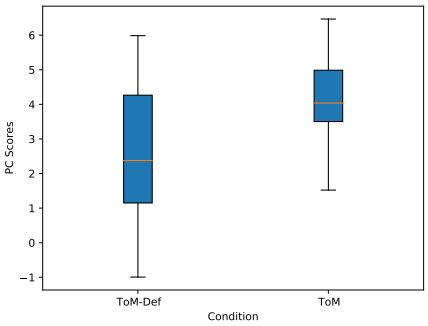

Fig. 3. Charger scenario - PC scores in the **ToM** condition were significantly higher than the **ToM-Def** condition ($p < .001$).

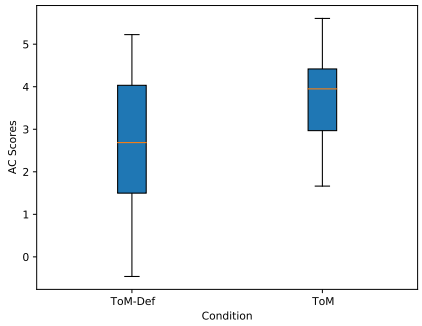

Fig. 4. Charger scenario - AC scores in the **ToM** condition were significantly higher than the **ToM-Def** condition ($p < .001$).

C. *Kitchen scenario*

1) Recognizes Human Cognitions (RC): Shapiro-Wilk tests showed a significance departure from the normality ($p < .05$) for both conditions. Therefore, we conducted a Mann-Whitney U test to determine if the perception of a robot being able to recognize the cognition of humans was different depending on condition. The differences between conditions were statistically significant with the **ToM** condition (mean rank = 53.13) being higher than the **ToM-Def** condition (mean rank = 28.49), $U = 307.000$, $p < .001$.

2) Predicts Human Cognitions (PC): Shapiro-Wilk tests showed a significance departure from the normality ($p < .05$) for both conditions. Therefore, we conducted a Mann-Whitney U test to determine if the perception of a robot being able to predict the cognition of humans was different depending on condition. The differences between conditions were statistically significant with the **ToM** condition (mean rank = 50.33) being higher than the **ToM-Def** condition (mean rank = 31.15), $U = 416.000$, $p < .001$.

3) Adapts to Human Cognitions (AC): Shapiro-Wilk tests showed a significance departure from the normality ($p < .05$) for both conditions. Therefore, we conducted a Mann-Whitney U test to determine if the perception of a robot being able to

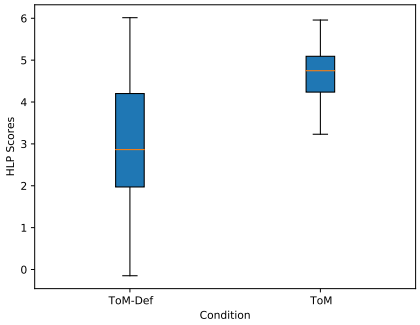

Fig. 5. Charger scenario - HLP scores in the **ToM** condition were significantly higher than the **ToM-Def** condition ($p < .001$).

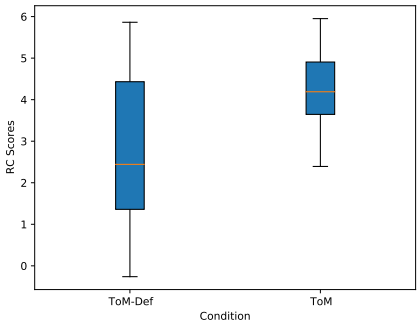

Fig. 6. Kitchen scenario - RC scores in the **ToM** condition were significantly higher than the **ToM-Def** condition ($p < .001$).

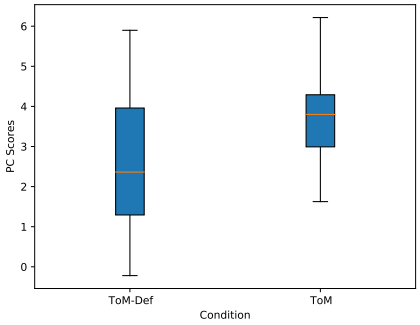

Fig. 7. Kitchen scenario - PC scores in the **ToM** condition were significantly higher than the **ToM-Def** condition ($p < .001$).

adapt to the cognition of humans was different depending on condition. The differences between conditions were statistically significant with the **ToM** condition (mean rank = 53.64) being higher than the **ToM-Def** condition (mean rank = 28.00), U = 287.000, $p < .001$.

4) Helpful (HLP): Shapiro-Wilk tests showed a significance departure from the normality ($p < .05$) for both conditions. Therefore, we conducted a Mann-Whitney U test to determine if the perception of a robot being helpful was different depending on condition. The differences between conditions were statistically significant with the **ToM** condition (mean rank = 55.37) being higher than the **ToM-Def** condition (mean rank = 26.35), U = 219.500, $p < .001$.

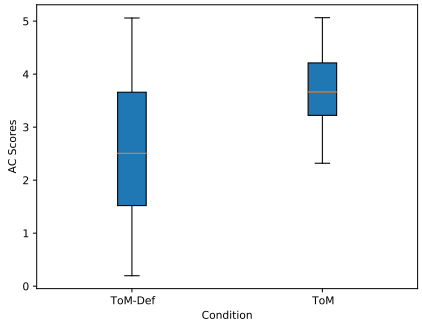

Fig. 8. Kitchen scenario - AC scores in the **ToM** condition were significantly higher than the **ToM-Def** condition ($p < .001$).

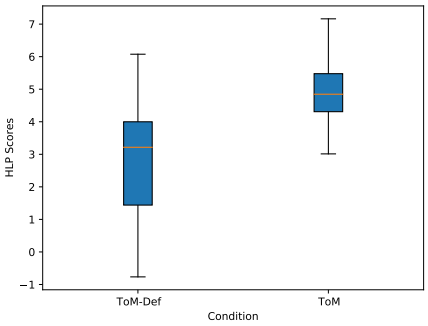

Fig. 9. Kitchen scenario - HLP scores in the **ToM** condition were significantly higher than the **ToM-Def** condition ($p < .001$).

D. *Corridor scenario*

1) Recognizes Human Cognitions (RC): Shapiro-Wilk tests showed a significance departure from the normality ($p < .05$) for both conditions. Therefore, we conducted a Mann-Whitney U test to determine if the perception of a robot being able to recognize the cognition of humans was different depending on condition. The differences between conditions were statistically significant with the **ToM** condition (mean rank = 52.23) being higher than the **ToM-Def** condition (mean rank = 29.34), $U = 342.000$, $p < .001$.

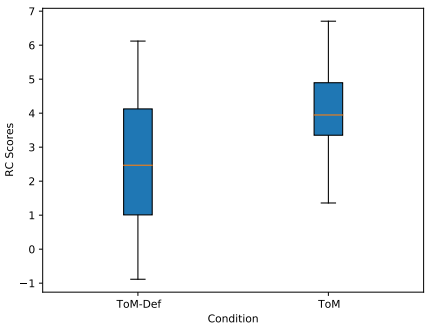

Fig. 10. Corridor scenario - RC scores in the **ToM** condition were significantly higher than the **ToM-Def** condition ($p < .001$).

2) Predicts Human Cognitions (PC): Shapiro-Wilk tests showed a significance departure from the normality ($p < .05$) for both conditions. Therefore, we conducted a Mann-Whitney U test to determine if the perception of a robot being able to predict the cognition of humans was different depending on condition. The differences between conditions were statistically significant with the **ToM** condition (mean rank = 51.23) being higher than the **ToM-Def** condition (mean rank = 30.29), $U = 381.000$, $p < .001$.

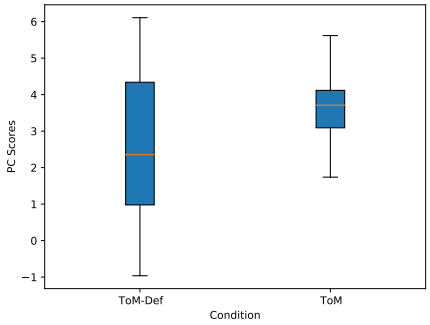

Fig. 11. Corridor scenario - PC scores in the **ToM** condition were significantly higher than the **ToM-Def** condition ($p < .001$).

3) Adapts to Human Cognitions (AC): Shapiro-Wilk tests showed a significance departure from the normality ($p < .05$) for both conditions. Therefore, we conducted a Mann-Whitney U test to determine if the perception of a robot being able to adapt to the cognition of humans was different depending on condition. The differences between conditions were statistically significant with the **ToM** condition (mean rank = 52.63) being higher than the **ToM-Def** condition (mean rank = 28.96), U = 326.500, $p < .001$.

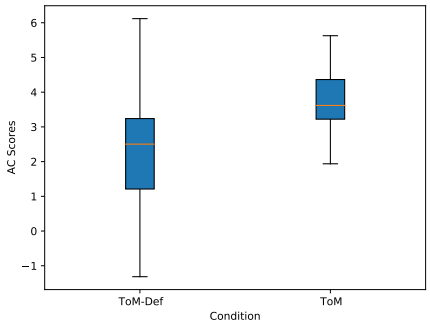

Fig. 12. Corridor scenario - AC scores in the **ToM** condition were significantly higher than the **ToM-Def** condition ($p < .001$).

4) Helpful (HLP): Shapiro-Wilk tests showed a significance departure from the normality ($p < .05$) for both conditions. Therefore, we conducted a Mann-Whitney U test to determine if the perception of a robot being helpful was different depending on condition. The differences between conditions were statistically significant with the **ToM** condition (mean rank = 52.13) being higher than the **ToM-Def** condition (mean rank = 29.44), U = 346.000, $p < .001$.

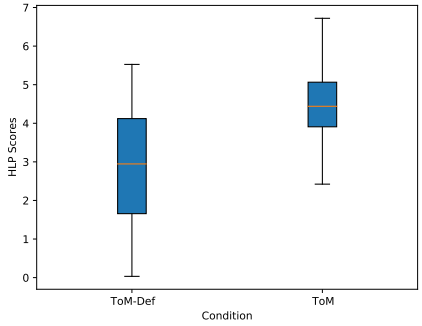

Fig. 13. Corridor scenario - HLP scores in the **ToM** condition were significantly higher than the **ToM-Def** condition ($p < .001$).

APPENDIX IX
DOMAIN DESCRIPTIONS

In what follows, we describe the various domains used in our evaluation of the robot's helpfulness discussed in Section VI.

A. *BW4T*

Johnson et al. [59] presented a multi-agent simulation platform, BlocksWorld for Teams (BW4T), which is an abstraction of a myriad of application domains such as search & rescue. Typically in this domain, there are a number of rooms and a drop zone, where each room contains a number of colored blocks. In the application domains, blocks may represent survivors of a disaster or medical kits, and the various agents may be humans or robots with different roles and capabilities. We cast blocks as medical kits.

Similarly to Shvo et al. [38], we modelled various instances of the BW4T domain by varying the number of rooms, medical kits, and types of medical kits, totalling 15 unique problem instances. The common theme to all problem instances is the following: there are two agents in the environment (Pepper and Bob); all discrepancies are perceived by Pepper and resolved by it; and all plans (except for the discrepancy resolving plans) are executed by Bob. Pepper can (truthfully) inform Bob of the whereabouts of various medical kits. Moreover, Bob can move medical kits between different locations in the environment. In all cases, Bob's goal is to get a particular medical kit to the drop zone. Bob's plan involves him going to the room in which he believes the medical kit to be. Pepper believes that Bob holds a false belief pertaining to the location of the medical kit. Recall that in the 'ToM + false beliefs' condition, Pepper holds false beliefs itself and may therefore provide Bob with incorrect information that will prolong Bob's plan.

Similarly to Shvo et al. [38], our BW4T domain was adapted from the BW4T domain in the RP-MEP repository[11] by reducing the number of actions found in the original domain.

Human replanning in our helpfulness experiments: In our helpfulness evaluation discussed in Section VI, following the execution of the discrepancy resolving plan in Line 11 of Algorithm 1, the simulated human agent's plan is executed. If it does not achieve the human's goal then RP-MEP generates a new plan for the human which involves sensing the environment until the goal is achieved. In the BW4T domain, this means that Bob searches all 'areas' of the room he is in and then proceeds to the closest room and searches all of its 'areas' and so on, until the medical kit is found. If Pepper provides Bob with information and changes his beliefs about the location of the medical kit, RP-MEP will generate a plan for Bob that involves heading to the location provided by Pepper.

B. *Corridor (Epistemic Planning Benchmark)*

In our modified[12] version of the *Corridor* domain, there are n agents in various rooms connected to a long corridor. A single acting agent, Bob, holds a secret and can move along the corridor, enter different rooms, and announce his secret. When announcing the secret, all agents in the room with Bob, as well as all agents in the adjacent rooms (when the door between the rooms is open), now believe the secret. The encoding of the *shareSecret* action in PDKBDDL is as follows:

domain-corridor.pdkbddl

```
(define (domain corridor)

    ; This specifies the agents in the set of agents Ag
    (:agents a b c)

    (:types loc door)

    (:predicates
        (secret ?agent)
        (at ?agent - agent ?l - loc)
        (connected ?l1 ?l2 - loc)
        (door_open ?d - door)
        (dummy)
    )

    (:action shareSecret
        :derive-condition   (at $agent$ l1)
        :parameters         (?a ?as - agent)
```

[11] https://github.com/QuMuLab/pdkb-planning/tree/ef396e686147bd3e2e392ffadebcefd07998f45f/examples/planning/bw4t

[12] The unmodified corridor domain can be found in https://github.com/QuMuLab/pdkb-planning/tree/ef396e686147bd3e2e392ffadebcefd07998f45f/examples/planning/corridor.

```
        :precondition       (and (at ?a l1) [?a](secret ?as))
        :effect             (and
                                (forall ?a2 - agent
                                    (when    (and (door_open dl1l2) (at ?a2 l2))
                                         [?a2](secret ?as))
                                         )

                                (forall ?a3 - agent
                                    (when    (and (at ?a3 l1))
                                         [?a3](secret ?as))
                                         )
                            )
    )

)
```

In our version of the corridor domain, Bob has the epistemic goal of selectively spreading his secret to the other agents in the environment. For example, let us assume that there are two agents in the environment in addition to Bob (k and l) and that Bob's epistemic goal is $B_k(\text{BobSecret}) \wedge \neg B_l(\text{BobSecret})$. That is, Bob wants agent k to believe his secret, but does not wish for agent l to believe it. We create 15 instances of this domain by varying the number of rooms, agents, and false beliefs held by Bob about the locations of each agent, and whether or not the doors between the different rooms are open or closed. For each of the generated instances, a tuple is created and given to Algorithm 1 where agent R is Pepper and agent H is Bob. Pepper can inform Bob of agents' locations and can also open and close doors in the environment.

We are interested in discrepancies pertaining to the validity of Bob's plan, as perceived by Pepper. Pepper will believe that Bob's plan is not valid while believing that Bob believes it is valid for the following reason:

- Bob only wants agent k to believe his secret without agent l believing it (i.e., $B_k(\text{BobSecret}) \wedge \neg B_l(\text{BobSecret})$). Pepper believes that Bob falsely believes that agent l is (1) neither in the adjacent rooms to the room in which agent k is located or (2) correctly believes that l is in an adjacent room but falsely believes that the door between the rooms is closed. Therefore, Bob will plan to go to the room in which he believes k to be and share his secret with her. However, this plan will fail to achieve Bob's goal since agent l will come to believe Bob's secret since the door between the rooms is actually open and l is in the adjacent room.

Human replanning in our helpfulness experiments: In our helpfulness evaluation, following the execution of the discrepancy resolving plan in Line 11 of Algorithm 1, the simulated human agent's plan is executed. If Pepper does not intervene and resolves the discrepancy it perceives between its beliefs and Bob's beliefs, then Bob's epistemic goal becomes irrevocably unachievable. This is because Bob will share his secret with agent k but will inadvertently also share it with agent l, because of his false belief. Consequently, $B_k(\text{BobSecret}) \wedge B_l(\text{BobSecret})$ will hold following Bob's plan and there is no action that can negate $B_l(\text{BobSecret})$. Therefore, Bob's epistemic goal cannot be achieved and so replanning will yield an empty plan.

C. *IPC Domains*

Inspired by 2 IPC domains[13] – Depots, Driverlog – we modelled 2 MEP domains with agents Pepper and Bob. In total, we generated 30 problem instances (15 from each domain) by creating false beliefs for Bob (e.g., causing Bob to hold a false belief about the location of an object in the Driverlog domain) and varying the domain parameters (e.g., number of objects in the domain). As before, Bob is the acting agent and Pepper is the discrepancy resolving agent.

In each IPC domain, Pepper has at its avail appropriate communicative actions. For example, in the Driverlog domain Pepper can inform Bob that he holds a false belief about the location of an object.

Human replanning in our helpfulness experiments: In our helpfulness evaluation, following the execution of the discrepancy resolving plan in Line 11 of Algorithm 1, the simulated human agent's plan is executed. If it does not achieve the human's goal then RP-MEP generates a new plan for the human which involves sensing the environment until the goal is achieved. In the IPC domains, this means that Bob searches all 'areas' of the location he is in and then proceeds to the closest location and searches all of its 'areas' and so on, until either the package (in the Driverlog domain) or the crate (in the Depots domain) is found, depending on the domain. If Pepper provides Bob with information and changes his beliefs about the location of the crate or package, RP-MEP will generate a plan for Bob that involves heading to the location provided by Pepper.

[13]IPC domains from multiple iterations of the competition can be found in `http://editor.planning.domains/`

D. **Home** *(domains used in our user study)*

In our study, we recorded videos involving a Pepper robot and a human(s) in a number of realistic scenarios: (1) a slight variation[14] of the kitchen scenario described in Section III, (2) a 'phone charger' scenario, and (3) a simplified corridor scenario (described earlier in Appendix IX-B). We also use these domains in our evaluation of the robot's helpfulness. In this section we elaborate on the phone charger scenario and the simplified corridor scenario.

1) ***Phone charger scenario****:* In the phone charger scenario, we place focus on *visual perspective taking* [50],[15][16] which is the ability to see the world from another person's perspective. In particular, we use *gaze detection* to recognize the direction in which each agent's gaze is facing. Recall that we use MonoLoco's orientation estimation and predict that a person has shifted their gaze if the predicted orientation changes. Previous work[17] has formalized visual perspective taking using epistemic logic and inspired by this formalization, we implemented a simplified visual perspective taking mechanism in RP-MEP [34].

Recall RP-MEP's conditioned mutual awareness mechanism that models and enforces conditions for agents' 'awareness' of the effects of an action. For instance, in our kitchen scenario agents are 'aware' that an action has been performed if they are in the same location in which the action was performed. To model visual perspective taking, we encode actions such that agents are 'aware' that an action has been performed only if their gaze is *facing* the 'appropriate' direction. For instance, if Bob is picking up an object in a certain corner of the room and Alice is facing the corner opposite to it, then Alice will not believe that Bob is now holding that object. With that in mind, we describe the 'phone charger' scenario and relate it to visual perspective taking:

> *Alice and Pepper the robot are in the living room. The scenario begins by Alice connecting her phone to a phone charger and immediately disconnecting it when she suddenly receives a call. Alice answers the call and proceeds to sit down at the table (with her back to the phone charger). As Alice begins her phone call, Bob walks into the living room, unplugs the charger, and leaves the room with it.*

After observing Alice connecting her phone to the charger, Algorithm 1's plan recognition component recognizes that Alice's goal is to charge her phone (and this goal persists even when Alice takes the phone call and sits down at the table). Algorithm 1's plan recognition component also recognizes that Alice's plan is to charge her phone using the charger.

As discussed in Section IV, the perception module in this case – upon observing that Alice sat down with her back to the charger – will produce the (informally[18] specified) observation *shiftGaze*(Alice, awayFromCharger). In Line 4 of Algorithm 1, the state S is progressed with the *shiftGaze* action, after which Pepper believes that Alice is not facing in the direction of the charger.

After observing that Bob has entered the room, unplugged the charger, and left the room, Pepper reasons that the charger is no longer there. However, via visual perspective taking, Pepper reasons that Alice – whose gaze is facing away from the phone charger – does not update her beliefs about the location of the charger.

Pepper now believes that Alice holds a false belief about the location of the charger (i.e., Pepper believes that the charger is no longer there while believing that Alice believes that it is still there) and therefore perceives a discrepancy between its beliefs and Alice's pertaining to her plan of charging her phone. Given that Pepper believes that there is another charger in the bedroom, in Line 8 an assistive solution π_{asst} is generated which involves Alice obtaining the charger from the bedroom. To resolve the discrepancies pertaining to the validity of Alice's plan and the assistive solution, Pepper can do one of two things:

> (1) Pepper goes to the bedroom, picks up the spare phone charger, and brings it back to the living room to replace the charger that was taken by Bob.

or

> (2) Pepper informs Alice that the charger is no longer in the living room and that there is another charger in the bedroom.

In our study, participants were shown the former discrepancy resolving plan executed by Pepper. The video can be found in `https://youtu.be/ICSKmchZIW8`.

Human replanning in our helpfulness experiments: In our helpfulness evaluation, following the execution of the discrepancy resolving plan in Line 11 of Algorithm 1, the simulated human agent's plan is executed. If it does not achieve the human's goal then RP-MEP generates a new plan for the human which involves sensing the environment until the goal

[14]due to the limitations of the perception system, we used a large bag chips instead of a can of soup and updated the PDKBDDL encoding, as well as the set of possible goal accordingly.

[15]Lakatos, G., Wood, L. J., Syrdal, D. S., Robins, B., Zaraki, A., & Dautenhahn, K. (2021). Robot-mediated intervention can assist children with autism to develop visual perspective taking skills. Paladyn, Journal of Behavioral Robotics, 12(1), 87-101.

[16]Gzesh, S. M., & Surber, C. F. (1985). Visual perspective-taking skills in children. Child development, 1204-1213.

[17]Gasquet, O., Goranko, V., & Schwarzentruber, F. (2016). Big brother logic: visual-epistemic reasoning in stationary multi-agent systems. Autonomous Agents and Multi-Agent Systems, 30(5), 793-825.

[18]More formally, the room was split into 4 quadrants and awayFromCharger specifies one of these quadrants.

is achieved. In the phone charger domain, this means Alice searches all 'areas' of the room she is in and then proceeds to the closest room and searches all of its 'areas' and so on. If Pepper provides Bob with information and changes his beliefs about the location of the phone charger, RP-MEP will generate a plan for Alice that involves heading to the location provided by Pepper.

2) ***(Simplified) corridor scenario****:* As mentioned, this is a simplified version of the corridor domain. In particular, the scenario is as follows:

> *Alice, Bob and Pepper are in the living room. Alice and Bob are sitting at the table with Pepper nearby, discussing work matters. Eve walks by and says goodbye to Alice and Bob. Bob whispers to Alice (with pepper within earshot): "I gotta tell you something about Eve's birthday party but Eve can't find out!". From Pepper's vantage point, it notices that Eve left the room but is right by the door (looking for something in her bag), within earshot but out of sight of Alice and Bob.*

As in the phone charger scenario, this scenario involves an element of visual perspective taking since Pepper believes that Alice and Bob's gaze is facing a direction such that they cannot see that Eve is right outside the door. As such, Pepper believes that Bob holds a false belief about Eve's location and therefore Pepper perceives a discrepancy pertaining to Bob's plan of sharing his secret with Bob. Pepper believes that if Bob shares his secret with Alice while Eve is just outside the door, Eve will come to learn the secret and Bob's epistemic goal will therefore not be achieved (and worse, Bob's goal will become irrevocably unachievable). The *shareSecret* action shown earlier in this section is slightly altered to model this.

To resolve this discrepancy, Pepper can either

(1) Send a message to Bob, informing him that Eve is standing just outside the door,

or

(2) close the door.

In our study, participants were shown the latter discrepancy resolving plan executed by Pepper. The video can be found in `https://youtu.be/VLxceOkYM-A`. In the video, Tara (playing the role of Bob) whispers to Ruthrash (playing the role of Alice) that she has to tell him about Maayan's (Eve's) birthday party but Maayan can't find out. In this work we do not address the interesting natural language understanding task of inferring Tara's epistemic goal from her utterance. Instead, Tara's goal is already known (and reflected in G_H) and given to Algorithm 1 – $B_{\text{Ruthrash}}(\mathit{Maayan\_birthday\_party\_secret}) \wedge \neg B_{\text{Maayan}}(\mathit{Maayan\_birthday\_party\_secret})$.

APPENDIX X
ADDITIONAL RESULTS – HELPFULNESS EVALUATION

A. An Infinitely Helpful Robot

In the last row of Table I, the cost of the plan π_H is always ∞ since the human's false beliefs cause her to generate an invalid plan that makes her epistemic goal irrevocably unachievable (e.g., Alice sharing her secret with Bob while Eve is, unbeknownst to Alice, behind the door eavesdropping). In contrast, the robot can assist by, for instance, informing Alice that Eve is behind the door, which causes Alice to replan and achieve her epistemic goal. The cost of the plan π_{HR} is therefore a finite number and, per [33], the robot's helpfulness in such cases is ∞.

B. Runtime results for Algorithm 1

While the focus of this work was not to evaluate the scalability of Algorithm 1, we discuss the runtime results from our helpfulness experiments (discussed in Section VI). The runtime and scalability of RP-MEP in conjunction with the epistemic planning-based techniques we leverage in this work have been evaluated by previous work [38], [45]. It has been shown that the runtime and scalability bottleneck is the depth of nested belief and number of agents in the MEP problem. These results are consistent with the findings of RP-MEP's developers [34] who discussed how RP-MEP's classical encoding of a MEP problem generates an exponential number of fluents which significantly impacts the runtime as the depth of nested belief and number of agents grow [20], [34]. In what follows we discuss runtime results for each RP-MEP-based component of Algorithm 1.

1) Epistemic plan recognition runtime: Shvo et al. [45] showed that when using RP-MEP in conjunction with a classical planner configured to generate satisficing (rather than optimal) plans, the runtime of their plan recognition algorithm remained low as long as the number of agents and depth of belief remained low as well. In Line 6 of our Algorithm 1, Shvo et al.'s plan recognition algorithm is called. The runtime results for our small domains (i.e., BW4T, Home, and Corridor) were similar to Shvo et al.'s [45] results – the human's presumed plan and goal were returned in under 1.5 seconds for all problem instances. This is due to the fairly small number of actions in the domain, small number of observations (due to short plans), and small number of agents. In contrast, the IPC domains Depots and Driverlog are more complex and plans generated to achieve goals in these domains are considerably longer than in our other domains. As shown in Shvo et al.'s Table 1 [45], runtime can reach minutes. Indeed, in our helpfulness evaluation we observed similar runtimes for the two IPC domains used.

2) Discrepancy resolution runtime: Shvo et al. [38] observed runtime results for their discrepancy resolution algorithm that align with those observed by Shvo et al. [45]. That is, as the number of agents and depth of belief grow, so does the runtime. Shvo et al. [38] even showed how, when the depth of nested belief was sufficiently high, the planner ran out of memory in some of the problem instances.

In Line 10 of our algorithm, we call a modified version of Shvo et al.'s discrepancy resolution algorithm. For the Home, Corridor, and BW4T domains, the discrepancy resolution algorithm returned a discrepancy resolving plan in under 2.5 seconds for all problem instances.

Similarly to the plan recognition runtime results reported earlier, in the IPC domains the runtime was considerably higher on average, with Line 10 taking up to 5 minutes to return a discrepancy resolving plan in some cases.

3) Real-time execution of Algorithm 1 in our user study video recordings: We note that RP-MEP's runtime can remain low when the number of agents and depth of nested belief stay low. This is observed in our simpler domains, as discussed in the previous subsections. Muise et al. [34] moreover mention that most interesting use cases of epistemic planning involve relatively low nested belief.

Indeed, our study included 3 fairly realistic scenarios that involve complex ToM reasoning. In our study, domains were simplified such that each iteration of Algorithm 1's while loop took less than a second. It was therefore possible to run the algorithm in real time and in conjunction with the perception module (i.e., the various perceptual detectors used).

4) Takeaway: While we were able to run Algorithm 1 in real time in our simplified domains, it is clear that in order to run Algorithm 1 in more complex domain and in real time, significant runtime improvements will have to be made. Fortunately, epistemic planning is an active research field and planner runtime will surely improve over time, as is the case for research on automated planning more generally.

APPENDIX XI
RELATED WORK

The related work section can be found in `https://bit.ly/3Ebb1tc`.