Probabilistic Planning for Symbiotic Autonomy in Domestic Robots

Nuno Mendes

Abstract— **Having robots interact with humans in domestic environments, while completing household tasks, requires a whole new degree of autonomy and reasoning. Typically, these kind of environments may push the robot's capabilities (i.e. robot has to pick an object on an unreachable location). Likewise, some actions performed by the robot agent have a higher success rate due to its domain knowledge. Cooperation between humans and robots can overcome these issues, resulting in a higher amount of possible tasks for the robot. This is an example of Symbiotic Autonomy.**

In this scenario, at some point the robot has the option to do some action by himself or to ask for help in case it's better for both agents - in the short or long-term.

The proposed approach uses a planning framework based on probabilistic logic programming, the HYPE planner, which gathers at every step, observations from the environment, generating a grounded Markov Decision Process problem from the described domain and deciding the action he should take in order to maximize its performance on this environment. Furthermore, this architecture is benchmarked on a simulation environment from generated observations as well as in a house with a robot and real human agents.

I. INTRODUCTION

It has been robotic's long term goal and also, dream, to have Social Mobile Robots and in particular Domestic Service Robots that can assist humans in their daily domestic tasks. This area is a subfield of social robotics which will likely have a considerable impact on society, not only by improving people's psychological and physical state, but also by giving them spare time which can be spent doing other activities. This includes robots: assisting senior citizens on tasks they can no longer execute on their own (like cooking, cleaning or organizing medical pills, for example); carrying out tasks for people with disabilities; and helping or dealing with house chores that are too cumbersome or repetitive for people.

Other rather important detail is that these DSR's often face several limitations on the actions they must perform and how they perceive the surrounding environment. Nevertheless, there are multiple ways to overcome these limitations, ranging from better sensors and actuators, to agents that seek help from other agents to reach their goals. However, the first solution is not always feasible: the robot could not withstand more weight; there is not a sensor precise enough for the application ; or more commonly, the budget does not accommodate any extra hardware/software to be integrated. Whereas the latter can be a cheaper and simpler solution for coping with these constraints. Under these circumstances, the robot agent should try to do most of the actions by himself, keeping its autonomy, but should ask for help whenever the success rate for a specific action is too low and the cost for asking human cooperation is bearable. This is an example of SA, where the robot has to take on tasks from people on the domestic environment, but can also ask for their help whenever it is necessary.

In general, the help that these robotic agents can receive is not restricted to physical actions, like grabbing a cup and giving it back to the robot, for example. Therefore, a human agent can also help the robot in many other ways, like giving him a better estimate of his position, giving him a preferable path to cross if there is some temporary physical restriction or by telling where is the object he is is looking for. Most of these pains are easily solved by these human agents, wandering around on these environments.

All things considered, by formalizing and solving this problem, the followed approach can be generalized to handle other kinds of domains where there is uncertainty in the world model and rewards for executing some action along with other human agents in the environment. **So, can symbiotic autonomy increase the robot's autonomy by improving the number of completed delivery missions on the domestic environment?**

II. BACKGROUND

A. Logical Programming

Prolog is a logical programming language first introduced by Alain Colmerauer and Philippe Roussel [1], in 1972, that was designed to simulate the man-machine communication system in natural language. The goal was also to integrate logic as a declarative knowledge representation language with a procedural representation of knowledge.

The ensemble of facts and clauses (in definitions 1 and 2, respectively) and queries are built out of terms, its single data type. Terms can be complex or simple, if they are composed by logical variables or constants (Atoms or Numbers). Variables start with an upper case letter while constants start with a lower case letter. As it is said before, what makes this programming language different than others is the fact that it is declarative, it does not describe the control flow of the program, only the logic of the computation. This logic is expressed by means of relations between objects, and its execution is started by running a query over the program's knowledge base. Programs are compiled by using a combination of facts and clauses which are in turn, transformed to a knowledge base (definition 3).

Definition 1 (Clause): Is a first order formula, represented by an implication clause that is read from the right to the left. The leftmost member is called the *Head* and the set of facts to the right are called the *Body*. This *Body* can contain both disjunctions and/or conjunctions of facts.

Definition 2 (Fact): Is a grounded (no variables) clause with an empty *Body*.

Examples of a fact and a clause are shown in snippets 1 and 2, respectively.

$$location(O, X) \leftarrow has(Y, O), location(Y, X). \quad (1)$$

$$robot(mbot). \quad (2)$$

Definition 3 (Knowledge Base): It is a collection of facts and relations which are loaded into program's memory, when it is executed.

B. Distribution Clauses

Distribution Clauses (DC) [1] [2] [3] are an extension of logic programming which can represent random variables as probability distributions. It includes typical probability distributions: *uniform*, *gaussian*, *discrete finite choice*, *poisson*, *beta*, etc. The formal syntax of DC's are normal *Prolog* definite clauses, but with few add-ons:

$$x \sim D \leftarrow b_1, ..., b_n. \quad (3)$$

Where x is the random variable, represented as a *Prolog* term; D is the probability distribution, which can also include other terms. Until now, every term explained is contained on the head of the clause. But DC's also include a body: $b_1, ..., b_n$, where b_k are literal terms. The body should have a valid substitution in order to generate variable x, and each grounding of the body will generate an instance of x. These special clauses can also include logical variables (when written in capitalized format) in both the body and the head, giving it an higher dimension of abstraction.

$$y \sim val(true) \leftarrow \simeq (x) = true. \quad (4)$$

After adding DC's into a *Prolog* program, it is now called a distributional program. The program will generate a distribution over possible worlds from the information given by all clauses in the Knowledge Base. The program works as follows:

1) Initialize possible worlds, W, as an empty set;
2) For every DC, find an unique substitution of the body and if it is possible, sample a value from the head's distribution and add it to W.
3) If it is impossible to generate more worlds, the program ends.
4) It is now possible to make probabilistic inferences from the sampled world.

Afterwards, probabilistic inference is made by querying the *Prolog* program over all possible worlds. An example of a distributed clause describing the robot location is shown in snippet 5. It models the robot location as a random variable and it is governed by a uniform distribution of possible values. On this case, the values are regions: kitchen, bed and sofa.

$$location(robot) \sim uniform(kitchen, bed, sofa) \leftarrow true. \quad (5)$$

C. Dynamic Distributional Clauses

Dynamic Distributional Clauses (DDC) [3] add more expressiveness to DC's by introducing temporal modeling. This is done by adding a subscript (as shown in snippet 6) to the previously defined random variables. This way, it is possible to describe logical interpretations that can evolve over time periods. In order to have this kind of dynamic clauses, random variables just need to have a time index associated with their program terms.

$$x_{t+1} \sim D \leftarrow b_1, ..., b_n. \quad (6)$$

In order to have a correct *Prolog* program with DDC, the random variables must be defined:

- in the initial timestep, $t = 0$, also known as the prior distribution.
- the transition model, which describes how a random variable evolves through successive time steps.

Example 5 can now be extended with the proper model, as seen in 7. So, the robot will always be in the same place as time passes.

$$\begin{aligned} location(robot)_{t+1} \sim val(X) \leftarrow \\ \simeq (location(robot)_t) = X. \end{aligned} \quad (7)$$

D. Markov Decision Processes

It is important to formalize a model that can capture the uncertainty contained on domestic environments with robots. These robotic agents must make optimal decisions (or approximately optimal) while knowing that their actions can possibly lead to multiple exclusive outcomes. A classical method is to formalize these domains as MDP's [4] [5] [6].

Definition 4 (Markov Decision Process): A process where an agent has to make decisions and the conditional probability of future states depends exclusively on the current state. It is characterized by the following tuple $< S, A, T, R >$:

- S, set of possible states.
- A, set of possible actions.
- T, state transition model.
- R, reward function.

[1]Implementation by Davide Nitti is available on `https://github.com/davidenitti/DC`

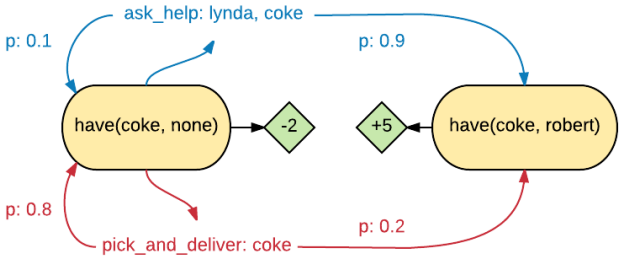

Fig. 1. Overview of a simple MDP for domestic environments, described for the robotic agent. There are only two states in this process, and they are colored yellow. Actions are colored blue and red. Outcomes of each action are enumerated from its name along with its explicit probability of occurring. Rewards are obtained when the robot reaches each individual state and are represented as green diamonds with respective values inside.

E. HYPE Planner

After a careful review of the available methods to solve MDP's, the choice fell upon HYPE algorithm [7], a state of the art planner completely written in *Prolog*. Some of its most important features include domains with an unknown number of variables and handling both discrete and continuous state and/or action spaces. All of this is accomplished after describing the domain, using DDC on a *Prolog* program, and following a set of syntax rules.

Unlike other classical algorithms based on dynamic programming, HYPE takes advantage of *Monte Carlo Learning* in order to overcome the curse of dimensionality and build the state-value function. Since others that follow this approach do not use information about the transition model, HYPE incorporates importance sampling in order to take advantage of that same model.

Also, and similarly to *Q-learning* [4], it follows an off-policy strategy to update the state value function. This means that the value of the optimal action is learnt without following the current policy. Its implementation uses an ϵ-greedy strategy to sample an action from the current state, i.e. with probability p chooses the action with the highest action-value estimate, otherwise, samples a random action according to an uniform distribution.

The algorithm takes advantage of logical interpretations for representing states in the domain. These logical interpretations are ground facts that define a possible world.

In order to write a syntactically correct HYPE domain, it is necessary to follow some rules:

1) Decide whether is preferable to use an implicit or explicit representation of the MDP. If an explicit model is chosen, there only exists one predicate at each timed step.
2) Identify needed predicates which are necessary to describe the MDP and goal predicate(s). Imagine, for example, that an agent has the goal of reaching the kitchen:

$$stop_t \leftarrow\simeq (location(agent)_t) = kitchen. \quad (8)$$

3) Enumerate all possible actions in the domain and explain when they are available to be used. Actions should describe what states they are applicable. A simple example is shown below, describing when the action deliver is available to be used:

$$\begin{aligned} applicable(deliver(coke, lynda))_t \leftarrow \\ \simeq (have(coke)_t) = agent, \\ \simeq (near(lynda)_t) = true. \end{aligned} \quad (9)$$

4) Choose the proper rewards: they can be function of specific interpretation 10, action 11 or even both 12. It can also be used to give a reward when the goal is reached 13. Though, it is important to mention that rewards are mutually exclusive. So, this can be a serious limitation if the domain designer chooses a factored state representation.

$$reward(10)_t \leftarrow\simeq (have(coke)_t) = lynda. \quad (10)$$

$$reward(-10)_t \leftarrow deliver(coke, lynda)_t. \quad (11)$$

$$\begin{aligned} reward(-4)_t \leftarrow\simeq (near(lynda)_t) = true, \\ deliver(coke, lynda)_t. \end{aligned} \quad (12)$$

$$reward(10)_t \leftarrow stop_t. \quad (13)$$

5) Describe the transition model of the domain using DDC's. It uses a structure similar to *Situation Calculus*. So, when some action is executed, it must dictate what will change and also what remains the same. An example is, for instance, the movement action of an agent and its effects on others around him:

$$\begin{aligned} location(agent)_{t+1} \sim finite([0.8 : hall, \\ 0.2 : kitchen]) \leftarrow action(navigate(hall)). \end{aligned} \quad (14)$$

$$\begin{aligned} location(lynda)_{t+1} \sim val(R) \leftarrow \\ action(navigate(Y)), \simeq (location(lynda)_t) = R. \end{aligned} \quad (15)$$

6) Instantiate a problem by grounding the current predicates with their respective values or probability distributions.
7) Try to find which hyperparameters (episode horizon, discount factor, exploration bias, number of episodes, etc) give the best domain results by differential parameter tuning under simulation benchmarks.

F. Symbiotic Autonomy

In robotics and similar to biology, when there is a symbiotic relationship, all agents are usually performing their own asynchronous actions, and each agent is influenced by the outcome of the other agents' actions. However, all agents can actively cooperate with each other, by communicating, requesting and providing help. As said earlier, agents with this type of relationship can ask or receive help from each other on actions they could not have performed by themselves, overcoming their limitations. Each agent can help the other one by: performing an action for the first agent; increasing the first agent's capability to perform an action.

By interacting in an environment with multiple human agents and possible actions usable by the mobile robot

agent, establishes this situation as a planning problem. So, each agent has a cost associated to his state and action made. Hence, each agent will perform the set of actions that minimize the cost of each others' state in order to achieve the goal.

It seems appropriate now to introduce the concept of *capability*: the probability of an agent to complete an action. If this probability is zero, it is impossible that this agent can successfully perform the respective action. Conversely, if the probability is bigger than zero, but lower than one, there is a chance that this action would not be completed by the agent. If the probability is equal to one, the agent can always perform this action successfully. This information can be encoded in the transition model of a MDP.

As it is not possible for a robot to be able to perform all actions in tasks that he is supposed to finish, due to the complex nature of the real world, this approach can be shown as an advantageous alternative to successfully complete most of the tasks assigned to the mobile robot agent. On the other hand, and as discussed before, this approach raises new problems related to probabilistic planning, as the models of the world become much more complex. This fact needs to be properly handled to give the best results possible in the current task. Furthermore, the cost of asking for help to a human agent must be accurately determined, and may even be different to each individual human agent [8], adding another level of difficulty to this problem.

1) Agent's Limitations: When a robot is wandering around an environment doing tasks. It is inevitable that it will experience a failure at some point, because of its limitations. There are a lot of possible causes that can lead to this outcome. Limitations can be divided in:

- **Action Execution:** the robot does not have a manipulator, or it will fail to grasp some object on a specific scenario. Another example could be its incapability to pass through stairways, or use an elevator.
- **Perception Errors:** the robot has difficulty identifying objects and people on a task. He could also have bad sensors which induce localization problems.
- **Insufficient Cognition:** these are the harder to overcome; An example is an agent that does not understand he will have to press a specific button in order to call an elevator.

2) Help Types: Hence, and in order to overcome the limitations described above the agent can request for help to people in the scene. So, the following types of help actions can be described:

- **Actor's Action Replacement:** the agent asks a nearby agent to perform some physical action for him. The human could pick an object and give it to the robot or he could lift him and carry it through a stairway.
- **Information Gathering Action:** the robot can ask for clarification about some uncertain measure which is relevant for its current state. An example of this behaviour could be a human correcting the robot's pose estimate on a map.
- **Policy Demonstration Action:** the human agent teaches the robot how he can interact in the environment in order to produce some effect in the world. This type of help action can augment the agent's capability in the long term, but with a higher cost. The human can teach the robot to touch the elevator button in order to get to another floor. He can also teach him to pick objects by demonstration.

3) Help Cost: Whenever the agent needs to make a decision in the environment, he must plan in advance if it is better to perform some action by himself or should he ask for help. The decision process takes into account each of the different costs to make the optimal decision. However, there are a couple of factors that should be taken into account when calculating the cost of requesting some human agent for help. Those can be:

- **Time the human agent takes to perform that action.** The time of a human is more valuable than the robot's. They can be mass produced and their purpose is to fulfill people's desires or to replace humans in repetitive and menial tasks.
- **Number of times the robot requested help to an agent.** A person will get tired of the robot soon if they are always receiving help requests. So each cumulative request to the same person gets more expensive to the robot.
- **Complexity of the help request.** The cost also depends on the type of help request. Help types were already defined in section II-F.2. The general rule attributes the highest cost to Policy Demonstration Actions and the lowest to Information Gathering Actions. However, there are exceptions and in those scenarios, a case by case study must be assessed.
- **Emotional state of a human agent.** Human agents are not always in the best mood to be requested for help. And the worst part is that emotions are normally hard to infer just by looking at a person. So, only by interacting with them (in a conversation, for example), they can discover more about their emotional state.
- **Personal's taste for robots.** Not all people feel the same about human robot interactions. There it will exist people that will ignore robots. Hence, the robot must be able to infer each human agent resistance to the interaction.

III. RELATED WORK

The prime example of the concept is project CoBot [9], in [10], where the mobile robot agent autonomously navigates between floors of CMU's Gates Center for Computer Science, escorts visitors to scheduled meetings and fulfills other needs which they might have. In this task, the visitor is not familiar with the building's layout and the robot is unable to do activities which might require physical manipulation of objects. Similarly, the robot could lose track of its exact location on the map. On the other hand, the human agent can easily manipulate objects around him and locate its exact position given a visual map, while the mobile robot can

plan optimal paths to multiple locations if its location is accurate. Given the previous conditions, the authors meshed the capabilities of each agent to overcome their limitations. They analyzed the convenience provided and its reliability to prevent delays on the visitor schedule while minimizing the help requests to other humans to increase its autonomy. Multiple policies executed by robot were analyzed by real world data. The domain description language used for this task was PDDL with probabilistic extensions.

Similar work was developed in efficient object search [11], by *Veiga et al*. They used probabilistic logic (this was accomplished while using *Problog*) to represent object locations as beliefs. Additionally, the object search decision module was modeled after a POMDP, in order to find objects in the minimum time possible. The information about the environment was stored in a semantic map and was updated repeatedly whenever it was acquired new data from sensors. That semantic map included probabilistic rules to update its knowledge about the environment. Tough, they used an offline POMDP solver in order to make decisions on the environment. Finally, they implemented their software pipeline into the *Mbot* platform. In contrast, this work focus on fully integrating PLPL into a domestic environment with possible human interaction (using symbiotic autonomy) in order to complete pick and delivery missions. The robotic platform for this task was the same.

IV. IMPLEMENTATION

In order to take advantage of MDP properties, the domestic environment was modeled taking into account uncertainty on robot's action effects. The task takes place in ISRoboNet@Home testbed which simulates a real domestic environment. The transition and reward models were obtained from empirical evidence that would come out of the HYPE planner. The domain is subdivided in two different domains, making up a hierarchical domain and simplifying the planning mechanism to produce better real world performance. The following case study tries to reproduce similar tasks that could happen on a domestic environment. Since the domains are implemented as *Prolog* code premises, its whole inclusion on this report would occupy too much space. So, the explanations made on this chapter focus more on a high level perspective of action models and sources of uncertainty. Nevertheless, the reader is advised to read the respective code that is included on the following repository `https://github.com/littlebrat/Master-Thesis`.

A. Hierarchical Overview

The designed task consists on a robot that wanders around the domestic household and carries out assigned tasks by the house's family members. The robot expects a reward every time it receives a mission or accomplishes the goal of an assigned task. The robot can only have one assigned mission at a given situation. Every single action the robot executes has a different given cost and can have multiple exclusive effects on the world. This effects are the result of stochastic processes, given the uncertainty in the real world. This schematic was successfully implemented by building an hierarchical model as seen on figure 2, in order to reduce the time it took to give correct results. The model was divided in two because the reward was awarded far into the future from the first step. So, the planner rarely got the expected action and would get caught in local maxima.

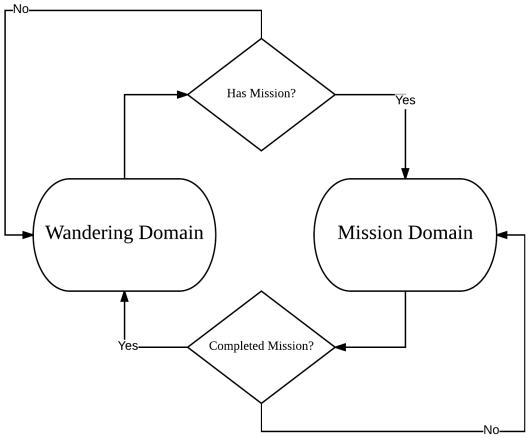

Fig. 2. Schematic of the domain architecture.

1) Wandering Domain Description: On this domain, the robot is wandering around the domestic environment while awaiting for requests given by other human agents. The robot can hear people calling for him (i.e. *"Robot come here!"*). Additionally, he is always aware of the location of each agent in the scene (including himself), and their position is static (apart from the robot). Additionally, when the robot moves to another room which has a person in it, there is a chance that he will be near that same person. Furthermore, the robot can engage in conversation with a person if the latter has called him earlier and is near him. While on conversation, the human may request an object, assigning a mission to him upon the respective validation. When a mission is assigned to the robot, the current working domain switches to the Mission Domain.

2) Mission Domain Description: The mission domain includes a pick-up challenge for the robot, assigned before by a human in the domestic environment. It includes a complete action description which enables him to pick and deliver the object himself or request help to another human while implicitly waiting to receive it. It also describes a special rule which penalizes requests for help to the same human that requested the object. It is important to mention that in our scenario a human cannot deny a request for help and its cost is the same for all available people in the house (apart for the human who requested the object in the first place). This behavior could be improved, but in this case, the description of the domain would have a higher degree of complexity and it would take a longer time to plan which was already a limiting factor for the current design. Similarly to the last domain, the location of every agent in the scene is known, including the object, though now every agent in

the domain can move from its original location. Finally, the mission will persist until the human who requested the object is in possession of it, switching to the Wandering Domain upon completion.

B. Software Architecture

The developed architecture (in figure 3) is composed of the following components:

- **Monitor Module:** receives external sensor data from multiple sources, processes and combines it with internal sensor data to form the joint state at each time step of the run. In order to procedurally accomplish this goal, the internal state is continuously tracked and modified according to the result of each action. Regular expressions were used in order to parse the external sensor data. The module is also responsible for generating the grounded MDP problem from the joint state and cleaning the previous one. When a problem is generated, it calls the next Module;
- **HYPE Module:** just a python wrapper for calling the HYPE planner with the desired domain and parameters: number of samples and episode depth.
- **Executor Module:** it receives the action to be performed from the previous module, parses it and sends a ROS action message to the respective action server. It was used a *movebase* server which was already integrated on the robot in order to send the robot to previously recorded waypoints on the map.

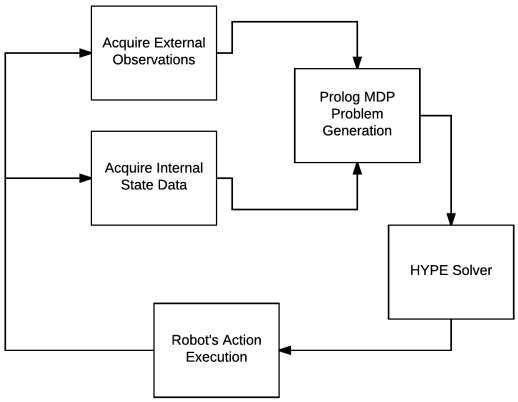

Fig. 3. High level diagram of the execution workflow.

On each cycle, the robot plans for a previously defined number of steps ahead, given the current observation data. So the current state and action space is given by the grounding of that same observation data. Though, from one step to another, the planning is done all over again. This is a critical limitation of this architecture and further work should focus on recycling the best policy obtained on the previous step.

In order to improve the performance of the developed module (the planner frequently causes CPU hogging), the workload was distributed between two computers on a single ROS network, using a remote master on the laptop which was responsible for running the **HYPE Module**, as shown in figure 4. The rest of the ROS network ran on the robot's on-board computer.

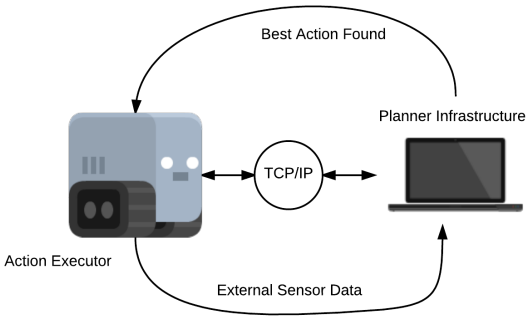

Fig. 4. High level overview of the communication between hardware systems.

V. EVALUATION

A. Physical Setup

1) Robot Platform: omni-directional 4-wheeled robot with a torso and rotating head. It is configured with two laser range finders (front and back of its base), Kinect 1 camera on the front of the head, plus an additional Asus Xtion PRO Live camera and a directional microphone Røde VideoMic Pro on top of its head for speech recognition. It also includes a robotic manipulator Robai Cyton Gamma 1500 which was not used due to hardware and software problems at the time.

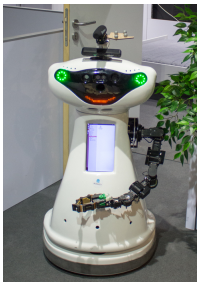

Fig. 5. *Mbot*, the robot that was used for the real world benchmarks.

2) ISRoboNet@Home - Home Environment Testbed: located in Instituto de Sistemas e Robtica, Lisbon. Serves as a benchmarking tool for robot's performance while on domestic environments. In figure 6, it includes a kitchen, bedroom, dining and living room, all fully furnished with commonly found appliances from *IKEA* store. This makes it easy to recreate similar environments on different laboratories and competitions across the world or in simulation environments, like *Gazebo*.

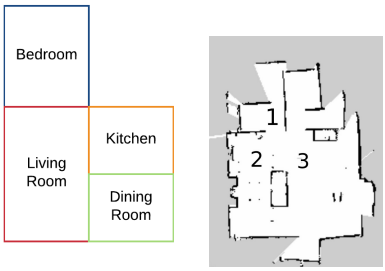

Fig. 6. On the left image, it can be seen a topological map of the ISRoboNet@Home testbed. On the right image, it is shown the map used by the robot for navigation and localization.

B. Simulation

First and foremost, all tests were performed on a single machine, equipped with an Intel i7 4700MQ CPU, 8 GB of RAM, on Ubuntu 16.04 KDE as operating system. The tests were performed on isolated environments with docker containers, so that they could be easily reproduced with minimum work on other computers or robotic platforms. The computer ran the tests while having the minimum amount of resources needed to operate. This means the computer was left alone to run these tests. Every benchmark test was executed 5 times to have a better estimate of its expected action choice, action-value estimates and its variance. Similarly, it was measured the time it took to return some output, with the same statistical estimators as before. It is also important to mention that a minimal number of tests were interrupted by the prolog's engine core dumps and so, their execution was ignored.

The description of the task consists of three parts:

1) On the first part, the robot is idle in the sofa location and the robot's goal is to find missions from nearby people. Suddenly, he is called by Robert. So, he should move to Robert's location (which is the bed). Afterwards, he should engage in conversation with Robert, in order to confirm the initial call and ask what he needs. At this point, Robert requests a coke beverage and the robot should acknowledge this request.
2) The second part relates to the mission domain where the robot needs to deliver the coke to Robert. At some point, he will have two mutually exclusive action choices: ask for help to get the coke; or grasp it and deliver it himself. But, in this case, the coke is already in the same location as Robert and the robot and, the latter is near it. So, the expected result should be the robot grasping and delivering the object himself, without any assistance.
3) The third part is similar to the previous one, but now the coke is in the kitchen table instead of the bed. So, the robot should move to the kitchen table location and request assistance to Lynda so that he can return to bed location and deliver the object to Robert.

C. Results

The purpose of these results was to test how the robot would behave given different scenarios, taking into account its own limitations and capabilities, and deciding appropriately when to ask for help or not. The other studied variable was the time it took to plan, in each step of the decision process. Results of the previously described simulation benchmark were obtained with 300 episodes/samples and a maximum of 10 steps on a running episode with $\gamma = 0.9$. The three following tables V-C, V-C and V-C represent each test referred before.

Step #	Best Action	Time (s)
1	wait	180
2	navigate(bed)	219
3	respond(ready_to_help, robert)	202
4	respond(robert, confirm_mission, want(robert,coke))	147

TABLE I

FIRST CASE STUDY RESULTS ON THE WANDERING DOMAIN.

Since the robot still does not have a mission he starts on the wandering domain. On the first step, the robot starts without having anybody calling him up for help. So, it decides the best action was to wait in the same spot instead of spending time wandering around the house, waisting energy. But, on the second step, the robot is called by robert which is in a different division of the house. It determines the best action is to leave the sofa in order to get to robert which is in the bed location. On the remaining two steps, the robot arrives at the bed division, standing close to robert, acknowledges the call and confirms the given mission request, after given the proper mission request from him. These results prove that the robot can receive help requests from human agents in the domestic environment. Talking now about the other variable: *time*. The time it took to plan was lower on the first and last steps. This happened because the planner had fewer states to explore. From the second step onward, the time it took to plan decreased as the number of possible states from the current one to the goal decreased. In each sample, the planner stops earlier if it reaches the goal condition.

Step #	Best Action	Time (s)
1	grasp(coke)	263
2	deliver(coke, robert)	109

TABLE II

SECOND CASE STUDY RESULTS ON THE MISSION DOMAIN.

The second experiment continues from the execution of the first one. So, the results shown here demonstrate the robot's behavior on the mission domain. Robot's mission is to deliver the coke can to Robert. The robot, Robert and the coke can are all in the bed location and the robot is near to others. On the first step, it had the possibility to execute multiple actions, including picking the coke can by itself or to move to other house divisions where it could ask for help to Lynda or Melanie to get the coke can. The robot determined that the best action would be to pick the coke can with its physical manipulator, despite its low accuracy.

On the second and final step, the robot delivered the coke can to Robert, finishing the mission assigned to it. This experiment was shorter, timewise, since it only took two steps to complete. The decreasing planning time trend is maintained.

Step #	Best Action	Time (s)
1	navigate(kitchen_table)	326
2	ask_help(coke, lynda)	292
3	navigate(bed)	264
4	receive(coke, lynda)	182
5	deliver(coke, robert)	105

TABLE III

THIRD CASE STUDY RESULTS ON THE MISSION DOMAIN.

The third and final scenario shows the robot's behavior on the mission domain, following the first experiment final state. The goal is the same as the previous case, but now the coke can is in another house division, the kitchen table. The robot and human agents are still in the same place. In the first step of this problem, the robot will have to make a decision similar to the previous scenario. It can go to the kitchen in order to grasp the object or ask for Lynda's help; or it can go to the sofa division to ask for Melanie's help. The robot's decision is to move to the kitchen. Afterwards, it decides the best decision is to ask for Lynda's help. Since the human agent behavior is also modeled on the domain, the robot knows how the human is expected to behave when he is helping the robot. So, on the following steps, the robot follows a policy which will culminate on it delivering the coke to Robert. Though, an unexpected behavior occured, as the robot left the room after asking for Lynda's help, making her run after it. This is domain design bug as it shows a greedy behavior from the robot's part. The planning time follows the same trend as on the previous case studies, decreasing with each step of execution.

D. Experimental Procedure

The experimental case studies were executed with help from people that were present in the lab at the time (test subjects). Then, test subjects would show QR codes to the robot which represented the external sensor information the robot would receive in a real scenario (as shown in figure 7). These physical experiments are just a proof of concept of the planned design, since sensor information was prefabricated. Tests were made to replicate the behavior which was obtained on simulation scenario. Finally, this experiment was filmed and can be found on `https://www.youtube.com/watch?v=Q3rLRO0bsqA`.

Fig. 7. Robot in a real world benchmark, on the mission domain situation, with a real human giving external sensor information with a QR code. On the left, the robot is in the sofa location and on the right the robot is in the bed location.

VI. DISCUSSION

A. Analyzing reliability of Symbiotic Autonomy on Domestic Environments

The introduction of SA on the robot's domestic task gives alternative action possibilities in order to accomplish some specific task on this environment. This translates into synchronous interactions with human agents. Human's action effects have an higher probability of leading to certain outcomes. This appears in contrast with the actions of the robot which sometimes lead to different states according to a probability function. However, this comes with the cost of possibly annoying the person being requested for assistance. Not only that but the amount of times it requests help to the same person are a reason of concern. These costs are taken into account in the description of the domain. Though, the relationship is not one sided, even if the robot asks for help, the human agent can also request help to get some object on the house. Looking into both sides, it is clear that there is a symbiotic interaction between the two agents where both benefit from helping each other. In short, robotic platforms can be more reliable if they trust some expert to execute some action instead.

Another important issue is how natural does the interaction between the human and the robot feel. Right now, the whole human behavior is predictable, deterministic and without any kind decision making taken into account. The reason is that the focus of this thesis was on having the whole system properly working, so there is room for improvement on human-robot interaction.

B. Looking into the High Level Architecture

When the modeling of the domain began, it turned out that the planner could not handle such a large domain. This meant that the planner would take too long to solve planning problems. To cope with this issue, a method was devised: organizing the domain into an hierarchical MDP. This way, planning domains are simpler in terms of possible actions and states, but the overall purpose of the system is maintained. This method provided a better response in terms of expected actions the robot would take for each encountered situation.

It is important to mention that external observations the robot got were simulated by QR codes. This happened because of time constraints of development. So an important step now would be to incorporate real world data into the system. Also related, the compilation of observations

was made based on predictable events released from the environment. This method was chosen instead of discretizing time into a set of steps, simplifying the perception module of the system. Though, in the future, it would be interesting to synchronize every sensor data into the ROS graph network.

C. Usability of the Dynamic Distributional Clauses

It is now possible to characterize the pains and gains of using this type of representation to describe MDP domains.

DDC's have plenty of potential to represent domains with higher degrees of complexity. They are able to represent predicates that change along time sequences and also variables distributed on a continuous space. Additionally, they offer common probability distributions to model these variables. It is also possible the usage of static functions for making external calculations that do not depend on the MDP problem. Equally important is that by taking advantage of a programming language, that is *turing complete*, as a way to model domains, makes for an higher level of expressivity that it is just not possible with other description languages.

Hence, looking into these gains, it would seem like a valid design choice to represent MDP's with this language, instead of using PPDDL or RDDL, for instance. However, when picking DDC, one must take into account the following drawbacks.

- One of the major negatives aspects of DDC is that it is largely inspired by situation calculus. This leads to the frame problem. For every predicate that remains unchanged after performing some action, a new rule has to be written. This quickly increases the amount of code written as the number of possible predicates and actions gets higher. As consequence, a lot of errors on the description of domains appear. Coupled with the high amount of rules written, it is really hard to track the origin of these errors, and the absence of debugging tools does not help as well.
- The implementation of this language was made on top of prolog. In fact, there are so many different implementations of prolog that in 1995 was published a standard to uniformize the structure of the language. Implementations that follow this standard are ISO/IEC 13211-1 compliant. As a result, the prolog programming community is largely fractured and declining over time. When some error occurs, it is generally hard to find a solution, as the problem could be related to a specific prolog implementation. DDC's is built with *YAP prolog*, a rather popular version. It is frustratingly common to have core dumps on execution of a program, making this language (in its current state) not proper for robotic applications. As rule of thumb, reliable robotic systems should rely on frameworks that have fast responses and easily recover from failures or unexpected events.
- This is not much of a drawback itself, but more of a discussion on why one should choose to describe planning domains with DDC's instead of PPDDL or RDDL. Originally, the formers came up in order to standardize planning domains, so researchers could focus on finding better planning algorithms, while having a common platform to benchmark them. On the other side, DDC is capable of describing domains in ways the others are not. It is the reader's job to interpret what is most important for their own planning research.

D. Is HYPE a good choice for solving real-time planning problems?

Withstanding the time constraints imposed by robotic's problems, HYPE planner provides a lot of flexibility by being able to dynamically introduce new predicates into planning problems as the domain changes. This includes new people or objects on the domain, for example. It performs decisions based on the current available information if the problem mantains the same knowledge base and structure. This feature is possible because it only finds a policy for the current state, and not for all the existent state space, since it is an online planner. One should not confuse online planning with real-time planning. The term online means that planning is performed on the moment it receives information from the environment. This appears in contrast with the majority of other planners which normally calculate the optimal policy for all possible state space, and that function is used on execution time, choosing the correct state-action mapping, if the policy is deterministic. This procedure is often called offline planning. Regardless, HYPE planner is not real-time since it does not guarantee a response within an acceptable temporal window or deadline.

Nevertheless, there are guarantees that given an infinite amount of time and the current state information, an optimal action is found. But, this is not good enough for robotics. An action solution should always be found on an acceptable task time. So, on its current state, HYPE is not an appropriate planner to deal with these type of problems, given its time constraints. Other planners should be considered when dealing with robotics domains that include some uncertainty on its domain, as *PROST*[2] or *POMDPX_NUS*[3] for instance.

Another important topic is the high amount of parameter tuning necessary to find the best action-value function. This happens because HYPE has a lot of hyperparameters, transforming the problem of planning into an optimization problem of finding the best parameter set for every specific planning problem. The high variance of the results also do not help on the task of finding them.

It is not possible to define rewards as the sum of multiple state factors. So, it is necessary to discriminate the reward for a complete world. This fact could easily lead to errors. Also, if there is world which can unify with multiple rewards, it will only be considered the first that appears on the text file. The solution for this problem was to exclusively use actions and goal states as possible reward sources.

[2] *PROST*, *Keller and Geisser*, is the winning planner from 2011/2014 ICAPS' International Planning Competitions, on the MDP boolean track.

[3] *POMDPX_NUS*, *Ye, Wu, Zhang, Hsu, and Lee*, won the 2014 ICAPS' International Planning Competition, on the POMDP boolean track.

Finally, other shortcoming of HYPE is that it can not model multi-agent domains. So, in order to model human agents in the domestic environment, it was assumed they had a static and known stochastic policy.

VII. CONCLUSION

On this work, it was proposed a framework for domestic robots using probabilistic logic programming with dynamic distributional clauses for solving MDP as one time decision problems. The robot uses these tools in order to decide which action to take at each time step while trying to maximize its expected reward on the domestic environment.

It is proposed an hierarchical MDP domain divided into two: one where the robot is wandering in the environment waiting for some request of assistance, and all actions are executed by this agent; and another where the robot has to complete some mission assigned from the first domain. This last domain provides the robot the chance to ask for assistance to another human in order to perform some action he had low probability of successfully carrying out.

The set of state predicates, applicable actions, rewards and rules of the domain are described in terms of dynamic distributional clauses. A problem is generated from these rules by grounding the observation set, given proprioceptive and exteroceptive information.

Finally, HYPE solver is called to solve this one time decision problem, giving as result the action with the highest state-action value, from the sampling procedure, along with the reward from executing that action. The robot parses the information from the program and executes the corresponding action on the real world while waiting to observe new information from it. This cycle continues until the goal of planning problem is reached, switching back again to the first domain where the robot is wandering around, looking for missions from human agents.

Nevertheless, the results obtained by the designed system were disappointing, since there is always one thing that fails throughout its operation and it takes a really long time to react to the environment.

VIII. FUTURE WORK

This thesis contributed to research in Symbiotic Autonomy between robots and humans in domestic environments while using a probabilistic programming language. However, if one wants to follow this line of research, it is really important to have fully working modules on the robot's own pool of hardware and software. Since, the developed work only made use of the robot's speech and movement, it would be interesting to replace the simulated actions of grasping and delivering objects with a real robotic manipulator that would react accordingly with the appropriate situation. Another important feature would be to incorporate real perception modules which can generate the expected observation data. It should include the following:

- **Speech Recognition**
- **Agent's Location Tracker**
- **Person Recognition**

All these improvements would make the benchmark much more realistic compared to a real domestic environment.

Talking now about one of the obvious shortcomings of the present software architecture, which is the time it takes to plan in any given situation. It would be an interesting research opportunity to integrate a state of the art planner into the developed module and compare its performance on a similar domain. I would propose PROST [12], given its records on past ICAPS-IPC's.

Another important shortcoming of the current architecture is the planner's incompatibility for modeling domains with multiple intelligent agents. In a real domestic environment, every agent is taking individual asynchronous decisions. A proposal for a research path would be to develop a new planner with similar characteristics to HYPE (i.e. using a similar programming language, and including DDC's), but with asynchronous multi-agent [4] decision making support for MDP's.

REFERENCES

[1] Alain Colmerauer and Philippe Roussel. The birth of prolog. In *History of programming languages—II*, pages 331–367. ACM, 1996.

[2] Davide Nitti, Tinne De Laet, and Luc De Raedt. Relational object tracking and learning. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 935–942. IEEE, 2014.

[3] Davide Nitti, Tinne De Laet, and Luc De Raedt. Probabilistic logic programming for hybrid relational domains. *Machine Learning*, 103(3):407–449, 2016.

[4] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[5] Dimitri P Bertsekas and John N. Tsitsiklis. Neuro-dynamic programming. *Athena Scientific*, 1996.

[6] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[7] Davide Nitti, Vaishak Belle, and Luc De Raedt. Planning in discrete and continuous markov decision processes by probabilistic programming. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2015.

[8] Stephanie Rosenthal and Manuela Veloso. Monte carlo preference elicitation for learning additive reward functions. In *RO-MAN, 2012 IEEE*, pages 886–891. IEEE, 2012.

[9] Manuela M Veloso, Joydeep Biswas, Brian Coltin, and Stephanie Rosenthal. Cobots: Robust symbiotic autonomous mobile service robots. In *IJCAI*, 2015.

[10] Stephanie Rosenthal, Joydeep Biswas, and Manuela Veloso. An effective personal mobile robot agent through symbiotic human-robot interaction. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pages 915–922. International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[11] Tiago S Veiga, Pedro Miraldo, Rodrigo Ventura, and Pedro U Lima. Efficient object search for mobile robots in dynamic environments: Semantic map as an input for the decision maker. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 2745–2750. IEEE, 2016.

[12] Thomas Keller and Patrick Eyerich. Prost: Probabilistic planning based on uct. In *ICAPS*, 2012.

[4] There is already a planning competition on these types of domains. See more about it here: `http://agents.fel.cvut.cz/codmap/`