Semantic Task Planning in Household Environments using Temporal Logic Synthesis

Author: Hyung Ju Suh

***Abstract*— In this work, the task planning problem in household environments is explored using linear-temporal logic (LTL). We present the task-planner as a hierarchical controller using two layers: 1) A discrete-space supervisory controller using LTL on semantically-abstracted states and control actions, 2) A lower-level interpreter that translates the semantic control actions and interact with the agent's physical-world dynamics. We justify the need for this hierarchy, and present the model for our framework. Finally, a household task-planning example is validated using our framework.**

I. Introduction

Robots for household services provide some of the most challenging problems for robotics, as they need to perform various general tasks within dynamic environments, cooperate with humans, and be safe to operate. Among these various challenges, two are emphasized in household environments: task planning, and interaction with humans. Robots in houses will need to be equipped with sufficient intelligence to plan complicated tasks that the human might request, whether it be fetching something, doing laundry, or cleaning the house. Additionally, interaction is a crucial problem, as everyday household users are unlikely to be equipped with the knowledge to program and understand the robot. As a motivational example, upon receiving a high-level command "get a beer from the fridge", the robot should be able understand the semantics of the natural language, and then plan on a sequence of actions necessary to complete the task. This work will particularly focus on the task planning portion of this problem.

Classical methods of task planning has been developed since the early days of artificial intelligence research, and have mostly relied on inference engines and formal logic programming. These include STRIPS [1], ADL[2], PDDL [3], and their derivatives. These languages define an initial state, a termination criteria, action schemas and their pre- and post-conditions, and the program handles these conditions to come up with a feasible task planning method. PDDL (Planning Domain Description Language) has seen excellent success in dealing with many household problems [4,5,6]. However, it has also been criticized due to its inability to address temporal constraints and resource constraints [7].

Recent alternatives to PDDL has been suggested from formal software verification, by observing formal logic and its connection to finite-automata. Most success has been seen from Linear-Temporal Logic (LTL), which is a logic specification able to handle temporal concepts. A great advantage of LTL-based methods are that the methods are verifiable. By exploiting the LTL-Automata equivalence [8], LTL formulas can be thoroughly checked for compliance with model checkers [9,10]. LTL methods has been successfully demonstrated for coarse Motion Planning [11,12], and multi-agent control [11,13]. Another advantage of LTL-synthesized controller lies in the fact that instead of using a linear-framework like PDDL, automatas can be made reactive by considering environment factors [11].

This LTL framework is later extended to address the problem of controlling hybrid dynamical systems, where the robot's inherent dynamics obey continuous-state properties, while the higher-level commands happen in discrete transitions. Such problems have evolved to take into account an agent's dynamical model along with the LTL specifications, enabling direct control of the agent's dynamics that is conformal to the given specifications. An example of such controllers is TuLiP [14]. These problems differ in objective from classic verification where a model is checked against given specifications, and instead synthesize a controller given a model and a specification.

Following such synthesis models, this work proposes a semantic formulation of a Linear-Temporal Logic (LTL) controller. In our work, we argue against controlling the agent's dynamics directly from LTL-synthesized specifications, and instead highly abstract the states and control actions using semantic labels. Combined with a semantic formulation of formal logic, we argue that this formulation is more tractable, efficient, and a step closer towards synthesizing high-level controllers directly from natural language.

The rest of the article is arranged as follows. In Section II, we discuss some limitations of current LTL-synthesized controllers and illustrate how they are ill-posed to handle certain classes of autonomy problems. These examples further motivate our framework which is formulated in Section III. Section IV contains a simple illustrative implementation of this idea. Finally, some limitations of this method and possible future works are mentioned in Section V.

II. Motivation

A. Background and Limitations of LTL controllers

In order to address some motivation for our framework, we mention some background and limitations of LTL-synthesized controllers. The LTL-controller is usually synthesized by combining two finite automatas: the first one represents the agent's dynamics as a finite transition system (FTS), and the second one represents the LTL specifications over a set of binary atomic propositions (AP). Combined, the resulting synthesized controller has a convenient advantage

of coming up with a scheme guaranteed to be compliant to both the agent's dynamics and the formal specifications [15].

To synthesize a system dynamics model and the LTL specification, a labeling function $L : \mathbb{X} \to 2^{AP}$ is applied to the system states in order to convert them into atomic propositions. In easier terms, this means the state-space of the agent is discretized into different regions. This partition process is also called state-abstraction, and is often done by a bisumlation algorithm [16,17] to satisfy the properties of the hybrid dynamics created by both the continuous state-space dynamics, and the discrete dynamics created by the specifications.

One major limitation of the LTL-synthesized controller lies in the resulting computational complexity. By discretizing continuous space, the controller must keep a physical representation of the world into all of its states in solving the synthesis problem. This has a direct scalability consequence that for problems that require higher resolution (near-continuous) or operate on higher dimensions, the computation becomes intractable [18]. Such an example could be mapping, navigation, and obstacle detection. An additional implication is that within the synthesized FTS, one loses access to operations and feedback directly from continuous space (without abstraction) in the decision making process.

Another limitation is that many existing LTL methods do not introduce the notion of optimality. This is indeed difficult to implement in a general LTL-formulation since various action inputs differ in what they are trying to achieve, and it is hard to imbue them with some associated cost or reward. For example, a motion planning problem might only care about the total traveled distance, while exploration problems might care more about how many moves it takes to explore the whole map. Thus it is inevitable that the notion of optimality must be specified within specific tasks the robot is trying to achieve.

B. Examples in Autonomy

To illustrate these points further, we provide examples of problems in autonomy that require a high resolution sampling, where a LTL-controller solution may be deemed ill-posed. Consider a cleaning (coverage) problem, where a robot must visit all the reachable spaces with a vaccuum cleaner. In a formal specification, it is convenient to say - "always eventually reach all the spaces", represented by the following LTL specification.

$$\varphi := \bigwedge_{i=1}^{n} \Box \Diamond c_n$$

where c_n corresponds to every discretized space within the work boundaries. The resulting controller would obey a recurrence relation, and visit all the states repeatedly. While this is a working solution, the first problem is that it is very difficult to partition c_n finely enough to cover all the spaces it must visit (in this case the partitioned space would be required to be smaller than the head of the vacuum cleaner). The second problem is that the order that the resulting controller decides to visit all these spaces are likely to be far from optimal (in this case optimality can be defined as shortest traveled distance required to cover the whole space [19]), as by the label function, there is no way to tell which states are physically closer to another.

Another problem is the exploration problem, where the robot is dropped at an unknown map, and must map the whole space efficiently. In an LTL specification, we can solve this problem by "if the location is unknown, reach the location next", or

$$\varphi := \bigwedge_{i=1}^{n} \left(b_n \Rightarrow \bigcirc c_n \right)$$

where b_i contains a boolean variable over whether or not the space has been observed, and c_i is an action variable that tells the robot to go to the space i. Let us say that one tries to introduce the notion of optimality by using a Maximum Expected Information Gain strategy [20]. Then, the algorithm is required to calculate the expected information gain of a partitioned space by summing over the expected information gain over its neighboring partitioned space. However, this algorithm is likely to suffer from the fact that the partitioned space does not have high-enough resolution to accurately portray the quantity being calculated.

C. Towards a Hierarchical Middle-Layer

The above examples illustrate the difficulty of incorporating the full-state dynamics of a given system and synthesizing it with a LTL specification, due to tractability of dealing with high-resolution partitioned space, and lack of optimality. One solution to this issue is to introduce a middle-layer between the system's dynamics and the high-level LTL controller, that is capable of the following: 1) Relaxing the computational load to the LTL controller by taking care of the agent's full-state dynamics, while indirectly abstracting the state-space to a tractable level. 2) Solving sub-problems of the abstracted actions optimally. The middle-layer would have access to operations in the continuous space, and also interpret abstracted commands from the LTL controller to interact with the continuous-space dynamics of the agent.

To illustrate this concept more clearly, let us visit the LTL solution to the animal herding problem in [11]. The continuous space is discretized through a simple low-resolution partition function that ends up in the following FTS in Fig. 1.

In the synthesized controller, upon deciding a location to reach, the robot plans the path by using a reachability algorithm in the FTS. However, two aforementioned problems arise: the abstracted space is too coarsely sampled for any reasonable motion planning, and the path is not guaranteed to be optimal. Thus we can leverage existing path planners such as Probabilistic Roadmap Method (PRM) [21], which can achieve path planning more efficiently and optimally. Fig. 2 illustrates the obvious comparison between the paths that the two methods might take.

For this simple case, one could argue that PRM is only an extension of the reachability idea by adding associated costs and running the shortest path search algorithm, and

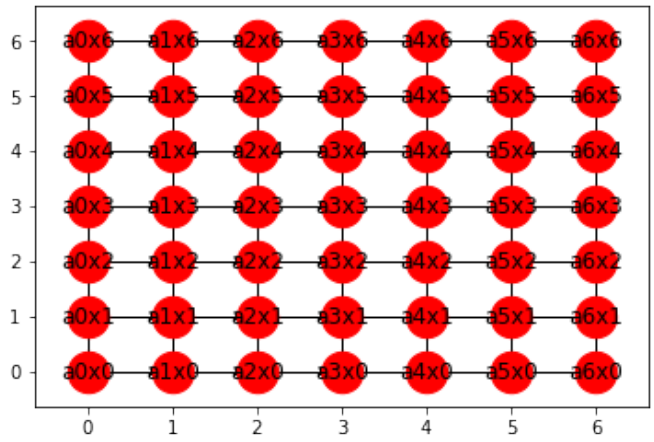

Fig. 1. FTS representation of discretized space map

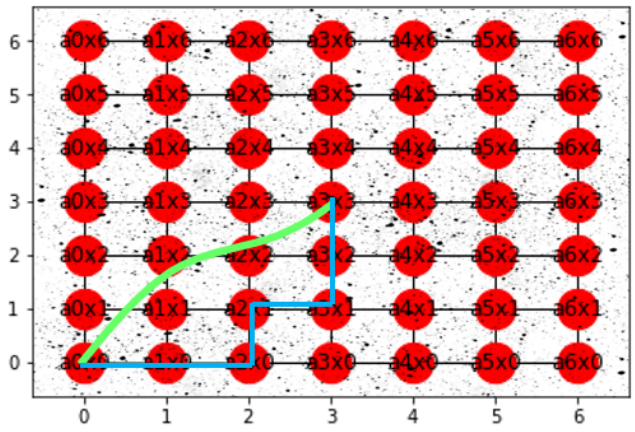

Fig. 2. Path from a0x0 to a3x3. Green is a path that an PRM with a reasonable resolution of sampling points might take. Blue is the path that the FTS might take.

such a feature may be implemented into the FTS. However, the general argument holds - for more complicated classes of autonomy problems like exploration or coverage, it becomes difficult and inefficient to synthesize a specification-based controller.

The role of the middle-layer would be to abstract these various operations through a keyword, understood as a discrete control input viewed from the LTL controller, and interpreted to a low-level control algorithm in the middle-layer. For instance, from the example of Fig.2, the LTL controller would issue a command "**move to** cell a3x3", and the lower-layer would run a PRM algorithm and communicate with the vehicle's low-level dynamics to achieve this goal. It is further possible to create additional hierarchies: for instance, there would often be a dedicated motion planner that listens to these commands and communicate with the hardware controller of the agent. More complicated algorithms that inherently require access to continuous space may be encoded this way as well - for example, "**clean**", "**explore**", or "**grab** the handle".

III. Semantic Framework

We extend the argument above to consider a semantic abstraction of the environment space, and the control action. The LTL-controller purely operates on semantically-labeled locations and transitions, and the middle layer interprets these semantic labels into mathematically well-defined operations that communicate with the agent's dynamics. In simple terms, locations will be encoded into words that have physical correspondence with real locations such as "**kitchen**", "**bedroom**", and actions will be encoded into words with semantically corresponding meanings, such as "**Grab**","**Move to**". Such a formulation has several conceivable advantages:

1) The supervisory controller is made indirect from the actual low-level dynamics of the agent and its operating environment, and only cares about high-level logic in a semantic level. This makes the decision process of the controller more tractable.
2) Each subproblems that arise from a semantically-labeled control action can be individually tackled optimally, subject to constraints that the subproblems might be interested in.
3) From a human-robot interaction perspective, it is easier to achieve better connection with natural language since natural language inputs may be limitedly parsed into commands and specifications for the LTL controller. Such methods are discussed in [22,23]
4) This process might be closer to how humans cognitively tackle task planning. Task planning happens at a semantic level through words, symbols, images, etc., while different subfunctions interpret the semantic commands and process them.

We model the detailed properties of this framework below.

A. Supervisory (High-Level) Semantic Controller

1) Finite Transition System (FTS): As mentioned before, the supervisory controller is a synthesis between two finite-automata. One automata models the discrete-dynamics of the system using a Finite-Transition System (FTS). In our case, the FTS is modeled as

$$\Sigma_S = (\mathbb{X}_S, \mathbb{U}, \delta_x)$$

where $\mathbb{X}_S$ is a set of semantically-labeled states, such as "**kitchen**". $\mathbb{U}$ is a set of semantic-labeled control actions, such as "**move to**", and $\delta_{\mathbb{X}}(x_s, u, x_s') = \mathbb{X}_S \times \mathbb{U} \times \mathbb{X}_S$ are transition relations on how the states evolve over a control input. An example might be a transition function described as δ(**'kitchen'**,**'move to'**,**'bedroom'**) which signifies that upon a control input '**move to**', the robot can transition from kitchen to the bedroom.

2) Semantic Abstraction Function: In order to describe how the semantically labeled discrete states $\mathbb{X}_S$ are mapped from continuous space $\mathbb{X}_C$, we define an abstraction function:

$$A : \mathbb{X}_C \Rightarrow \mathbb{X}_S$$

An important property of this function is that the function is injective but non-surjective, meaning that a single element in continuous coordinates $\mathbb{X}_C$ can corresponds to multiple elements in the semantic space $\mathbb{X}_S$. A simple example would be a coordinate in the refrigerator that also belongs to a kitchen. This differs from partition functions using bisimulation, which guarantees bijective properties between continuous space and the partitioned discrete space.

Potentially, this is problematic on two fronts. First, according to this function, an agent may belong to multiple states at once. If such an issue occurs, we define this function to map to the smallest set such that given the current continuous coordinate x_c, we find the set x_s that contains x_c and has smallest area. In $\mathbb{R}^2$, this may be calculated as

$$x_s = \min\Big(\int_X x_c dA, X = \{x_c | A(x_c) = x_s\}\Big)$$

The second problem is that it becomes ambiguous to define the inverse of this function $A^{-1} : \mathbb{X}_S \Rightarrow \mathbb{X}_C$ that can recover continuous coordinates from a semantic element. For instance, upon receiving the transition δ(**'kitchen'**,**'Move to'**,**'bedroom'**), we want to run a motion-planner with initial position as kitchen, and final state in the bedroom. However, the motion-planner will need to know exactly where (a continuous coordinate) in the bedroom it needs to go to. We solve this problem by arguing that the exact coordinate does not matter as long as it can be mapped back to the semantic location. This can be seen from how humans tell others to go to a location using the name of the location, and not GPS coordinates. Locations that require more precise coordinates will have a smaller set that maps to its name, such as "**near the Refrigerator**","**near the Dinner Table**", etc.

Additionally, we comment that one can also leverage the set relations between elements of $\mathbb{X}_S$ to aid in task planning. For instance, the robot will have to visit the kitchen first before it can access the refrigerator, and open the second container in the bottom, which may help it set waypoints for its goal. This idea is explored in [24].

3) LTL Specifications and Synthesis: Another finite automata deals with a set of specifications through atomic propositions, and is a Deterministic-Finite Automata (DFA)

$$\Sigma_Q = (\mathbb{Q}, 2^{AP}, \delta_q, q_0, Q_f)$$

where $\mathbb{Q}$ is a set of automated states, AP is a set of logical atomic propositions used, $\delta_q(q, ap \in 2^{AP}, q') = \mathbb{Q} \times 2^{AP} \times \mathbb{Q}$ is a transition relation between each state, q_0 an initial state, and $Q_f \subset \mathbb{Q}$ a set of acceptance states. For instance, a LTL specification may hint that if the refrigerator is open, it should be closed. Then the state representing the refrigerator door will behave accordingly.

The rough synthesis of these two finite automata is

$$\Sigma_S \otimes \Sigma_Q = (\mathbb{X} \times \mathbb{Q}, \mathbb{U}, \delta_x \times \delta_q)$$

where $\delta_x \times \delta_q$ allows a transition from $(x,q) \xrightarrow{u,ap} (x',q')$ if and only if $\delta_x(x,u,x')$ and $\delta_q(q,ap,q')$. While this is illustrative for an idea behind synthesis, this method only applies to a subset of LTL formulas called syntactically co-safe Linear Temporal Logic (scLTL). The general LTL formula requires translation into a Rabin-automata, which may be synthesized in a similar way.

Additionally, we comment on the labeling function $L_{\mathbb{X}} : \mathbb{X}_S \rightarrow 2^{AP}$. In this case it is quite straightforward to implement since we already have discrete states, and the membership in each state can be represented by an atomic proposition.

4) Action & Object Representation: In order to represent objects and actions, we leverage the fact that because the actions ($\mathbb{U}$) and objects ($\mathbb{O}$) are semantically-represented and therefore discrete, they can also be turned into binary atomic propositions by similar label functions

$$L_{\mathbb{U}} : \mathbb{U} \Rightarrow 2^{AP} \qquad L_{\mathbb{O}} : \mathbb{O} \Rightarrow 2^{AP}$$

Thus, one can simply take these atomic propositions and synthesize them into LTL specifications.

One can also argue whether or not it is necessary to record not only the binary detection of objects, but also the location of each object within the semantically-labeled space ($\mathbb{X}_S$). It is possible to achieve this by keeping record of where each object is in, and combining the two label functions $L_{\mathbb{X}} \times L_{\mathbb{O}}$. However, we argue against this method due to the resulting significant increase in DFA memory by $|\mathbb{X}_S| \times |\mathbb{O}|$, and also due to the fact that the objects' locations are highly dynamic, with human agents within the environment. An alternative is to use a semantic knowledge database [25,26,27] to infer a likely place to find an object (a beer is likely to be in a fridge), or to request input from the human.

5) Passive Transition: Finally, we mention an interesting consequence of our framework - because the actual dynamics of the agent is controlled by the low-level controller, there is no guarantee that the transition will deterministically lead to its desired state. As an example, imagine a controller issues a command "**clean**" when the robot is in the bedroom. Then the robot moves from state-to-state ("**bedroom-bathroom-kitchen**"), but this transition is now controlled by the low-level controller, as opposed to the LTL transition function. Thus a 'passive transition' occurs.

To resolve such passive transitions, there are two solutions. If we have a priori knowledge that the commanded action will display such behavior, we can add this transition function between every two states in $\mathbb{X}_S$, and only look at the start and end states of the low-level controller. This would make sense for control inputs such as **move to**. Another solution is to make a self-looped transition, and force the lower-level command to return to its original position once the action is terminated. This might make sense for control inputs such as **explore** or **clean**.

B. Low-Level Action Interpreter ("The Middle Layer")

The second component of our framework is the low-level controller that interprets semantic command outputs from the high-level controller $u \in \mathbb{U}$, and interacts with the agent's dynamics. In other words, this layer contains a declarative

database of functions that are required to specifically carry out the commanded task.

1) Action Primitives: We can see that the semantic control actions $u \in \mathbb{U}$ act like action schemas in PDDL. In our work we label them as action primitives. We illustrate some high-level examples of how specific commands might be interpreted to specific actions in this database.

- **Move to** ⟨**location**⟩: Upon receiving a semantic location $x_S \in \mathbb{X}_S$, we choose a reasonable continuous coordinate such that $A(x_C) = x_S$. (For example, centroid of the set $\{x_C | A(x_C) = x_S\}$). Then given its initial coordinate $x_{C,i}$ and goal coordinate x_C, these parameters could be passed on to a generic motion planning method. An example of such a motion planning method is path planned by PRM, and a closed-loop low-level motion controller (for instance, PD control of motion based on measurements from localization) to follow this path.
- **Grab** ⟨**object**⟩: Upon receiving a semantically labeled object $o \in \mathbb{O}$, the robot can perform visual template matching / classification to identify this object [28], and do a rough 3D reconstruction [29]. Then the object's 3D voxel coordinates and the robot's current position can be passed on to a grasp planner to grab the object [30,31].
- **Find** ⟨**object**⟩: Upon receiving a semantically labeled object $o \in \mathbb{O}$, we can ask the robot to visually cover all coordinates x_c, and run template matching to locate the object's coordinates.
- **Open/Close** ⟨**container**⟩: this is a rather complex problem, that must be solved with access to continuous coordinates. Once the robot finds the door and its handle through template matching, it can utilize methods in [32]
- **Clean** : This is a coverage problem, where the robot will find the shortest path that can cover all coordinates in continuous-space, and follow this path using a motion controller [19]. Upon termination the robot can return to its original location.
- **Explore/Map** : One may implement optimal strategies to find the maximum expected information gain [19], until a termination criteria is met and the robot explores the whole map.

Notice that many of these actions cannot be broken down further without sacrificing optimality and simplicity in their own domains. Thus the action set $\mathbb{U}$ can be considered as action primitives for a robotic agent, that the higher-level controller has access to in order to interact with the world. One can choose to add other basic action primitives such as **drill**, **run**, or **make a noise**, or more complicated sets of actions such as **do laundry**. A distinction (and a limitation) of whether or not a fairly complicated action should be included in the action set $\mathbb{U}$ can be answered in two fronts:

1) If the action requires its own notion of optimality, it should be included in the action set.
2) If the action requires access to continuous-space, it should be included in the action set.

Finally, we comment that individual actions may run their own logic-synthesized controllers, which are required to handle inherently complex and hybrid-dynamic problems such as folding laundry, or grasping [32].

IV. Example & Implementation

To illustrate our framework, we implement a semantic LTL controller using TuLiP [14], and simulate a household environment with few relevant locations and objects. For the sake of convenience, heavy robotic tasks such as manipulation and grasping is left out, and the robot is abstracted to be able to do such complex activities as long as it is in its action-primitive set $\mathbb{U}$. The household environment is illustrated in Fig.3, with door elements and obstacles such as sofa, bed, and TV.

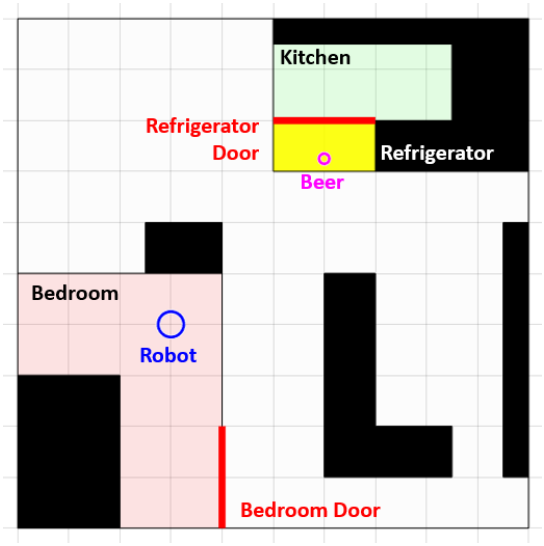

Fig. 3. Household environment used for the example. The bedroom is colored in red, kitchen in green, and the refrigerator in yellow.

The performance of the semantic LTL controller is displayed with a corresponding physical-world movement, upon given a task "**Bring a beer from the fridge**". Upon successful execution of this task, the robot is expected to figure out the required following steps in Algorithm 1.

Algorithm 1 Answer to: **Bring a beer from the fridge**

1: Start at the bedroom
2: Open the bedroom door
3: Go to the Kitchen
4: Go to the refrigerator
5: Open the refrigerator door
6: Grab the beer
7: Move to the bedroom

A. Variable Setup

To set up our simulated environment, we define the relevant variables. In the notation of TuLiP[14], and [11], the variables are divided into environment variables (sensor propositions) and system variables (robot actions), where the environment variables $\mathbb{E}$ is something beyond the robot's

control, and the system variables $\mathbb{S}$ are actions to be taken by the robot. First, we define our notion of actions, places, and objects:

$$\begin{cases} \mathbb{U} & \textbf{Move to, Open, Grab} \\ \mathbb{X}_{\mathbb{S}} & \textbf{Kitchen, Bedroom, Refrigerator} \\ \mathbb{O} & \textbf{Beer} \end{cases}$$

Then we can first set up environment variables for our objects:

$$\mathbb{E} = \{o_{\text{beer}}\}$$

where this variable will be set to true if the robot has the beer, and false if the robot does not. Next, the system variables are defined as below:

$$\mathbb{S} = \{ \begin{cases} r_{\text{bed}}, r_{\text{kit}}, r_{\text{stay}} & \textbf{Move to} \\ op_{\text{bed-kit}}, op_{\text{ref}} & \textbf{Open} \\ g_{\text{beer}} & \textbf{Grab} \end{cases} \}$$

The number of system variables required are directly related to the size of $\mathbb{U}$, and what it applies to in $\mathbb{X}_s$ and $\mathbb{O}$, and worst-case maximum size is $|\mathbb{U}| \times (|\mathbb{X}_s| + |\mathbb{O}|)$, if we assume every action can be applied to every location and object. The semantic FTS describing describing our state-system is shown in Fig.4.

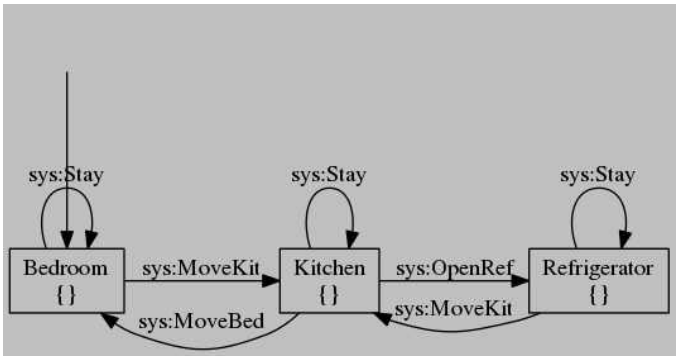

Fig. 4. FTS showing the states and transitions

B. LTL Specifications

Next, we describe the LTL specifications for the setup. The specifications are divided into progression specifications and safety specifications in TuLiP, which are subject to different temporal operators. For our semantic LTL, we interpret the difference between the two as follows: progression specification portrays intent on how to interact and manipulate with the world, while the safety specification portrays declarative knowledge about the rules in the world. As a simple example, the move to the kitchen from the bedroom is a progression specification. However, the fact that you cannot pass through it before you open the door first is a safety specification.

The implication of this interpretation is that only the specifications describing the progression needs to be changed upon given different task commands, because the safety specifications are only a collection of declarative knowledges that must be obeyed at all times. Thus if we have a comprehensive database of declarative specifications about the household environment, the robot simply needs to re-synthesize a new controller with new progression specifications. If the controller is unrealizable, the robot may then report back to the human. We discuss some possibilities of auto-generating such progression-specifications in Section V.

For our case, the specifications are written below in words:

1) Progression Specifications
   a) When not grabbing the beer, go to the refrigerator
   b) When in the refrigerator and not grabbing the beer, grab the beer
   c) When grabbing the beer, come back to bedroom
2) Safety Specifications
   a) If the bedroom door is closed, the agent cannot go from the bedroom to the kitchen.
   b) If the bedroom door is closed, the agent cannot go from the kitchen to the bedroom.
   c) If the refrigerator door is closed, the agent cannot grab the beer.

In formal logic, these specifications are written as follows:

$$\varphi := \begin{cases} \bigwedge \square \lozenge \left(\neg o_{beer} \Rightarrow op_{\text{ref}}\right) & 1.a \\ \bigwedge \square \lozenge \left(o_{beer} \Rightarrow r_{bed}\right) & 1.b \\ \bigwedge \square \lozenge \left((op_{\text{ref}} \wedge \neg o_{beer}) \Rightarrow g_{beer}\right) & 1.c \\ \\ \bigwedge \square \left((\neg op_{bed-kit} \wedge r_{kit}) \Rightarrow \neg \bigcirc r_{bed}\right) & 2.a \\ \bigwedge \square \left((\neg op_{bed-kit} \wedge r_{bed}) \Rightarrow \neg \bigcirc r_{kit}\right) & 2.b \\ \bigwedge \square \left((\neg op_{ref} \wedge r_{kit}) \Rightarrow \neg \bigcirc g_{beer}\right) & 2.c \end{cases}$$

Finally, we simulate the environment such that when the robot grabs the beer in the refrigerator, then its sensor variable for the beer turns true. $((g_{beer} \wedge op_{ref}) \Rightarrow o_{beer})$. The synthesized controller is given in Fig. 5, but due to lack of resolution, a more detailed image can be seen in this link.

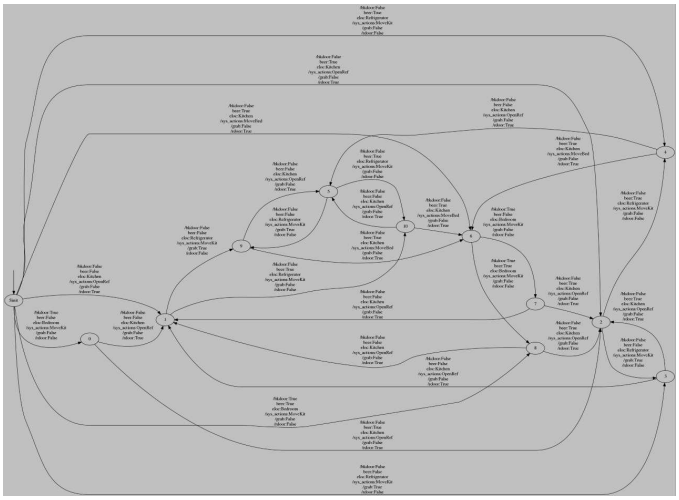

Fig. 5. Synthesized controller with the FTS in Fig.4 and the LTL specifications

C. Simulation

The behavior of our synthesized controller is verified with a simulation on the household environment in Fig.3. The results are shown in Fig.6. In implementation, the **Move to** action-primitive is carried out by a PRM algorithm, and the samples are overlayed with the environment to illustrate the resolution. A similar idea is implemented in [34]. Other abstracted action primitives can be implemented as well, creating a real-life simulation or experiment.

As shown from Fig.6, we can observe that the controller carries out all the necessary actions as described from Algorithm 1, showing a successful implementation of a task planner. However, because we chose to use a PRM algorithm for motion planning instead of labeling the whole space, the solution is much more tractable and optimal. The coarse sample in Fig. 5 contains 6312 samples, and in order to achieve the same resolution by partitioning using bisimulation properties, the resulting FTS would quickly become intractable.

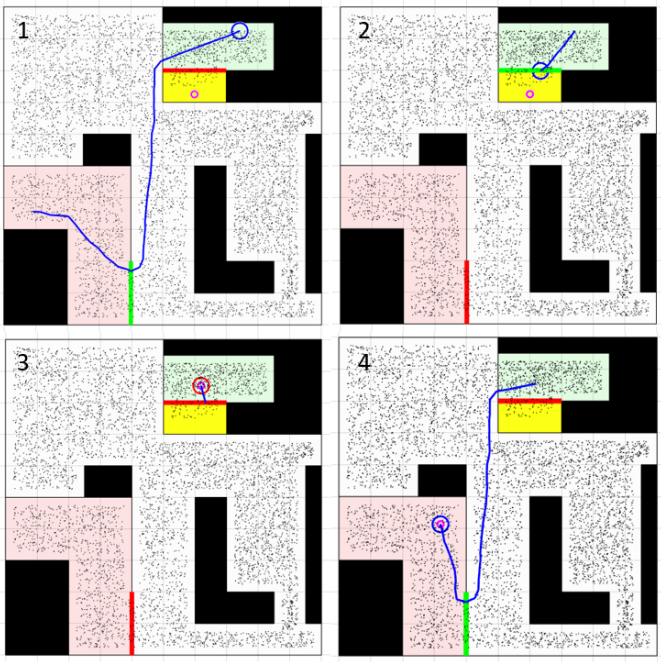

Fig. 6. Simulation-verified behavior of the synthesized controller on our simulation environment. The doors are red when closed, and green when open. 1. The robot opens the bedroom door and goes to the kitchen. 2. The robot opens the refrigerator and goes near it to access it. 3. The robot grabs the beer from the fridge and goes back to the kitchen, closing the refrigerator door. 4. The robot opens the bedroom door again and goes back to the bedroom.

We also comment that the semantic task planner using LTL is more robust against dynamic environments, compared to linear task-planning methods such as PDDL. In Fig.6, we can see that although the robot opened the door while going from the bedroom to kitchen in Fig.6.1, the door becomes closed again when it comes back to Fig.6.3. If we interpret this as a human interference within the environment, then the controller will be able to take into accounts these dynamic variables and adjust its action accordingly [35].

D. Simulation: Error Recovery

We argue that since the robot adjusts its actions through sensor conditions, it can perform well in recovering from possible errors that the robot might experience through failure of one of the lower-level action-primitives, or due to environment intervention. This is analogous to a feedback structure where the higher-level controller commands something, and the lower-level interpreter executes it, but the higher-level controller can get feedback by sensor variables.

Fig.7 shows such an example where the robot takes the action to grab the beer (Fig.7.3), but fails to grab the beer because a human has accidentally bumped into it, or the manipulation subproblem has failed. This is simulated with forcing the beer sensor variable to false when the robot grabs it. The robot than alters its behavior to try to grab it again, and reaches for the refrigerator (Fig.7.4). When it finally succeeds (Fig.7.5), it returns to the bedroom. (Fig.7.6)

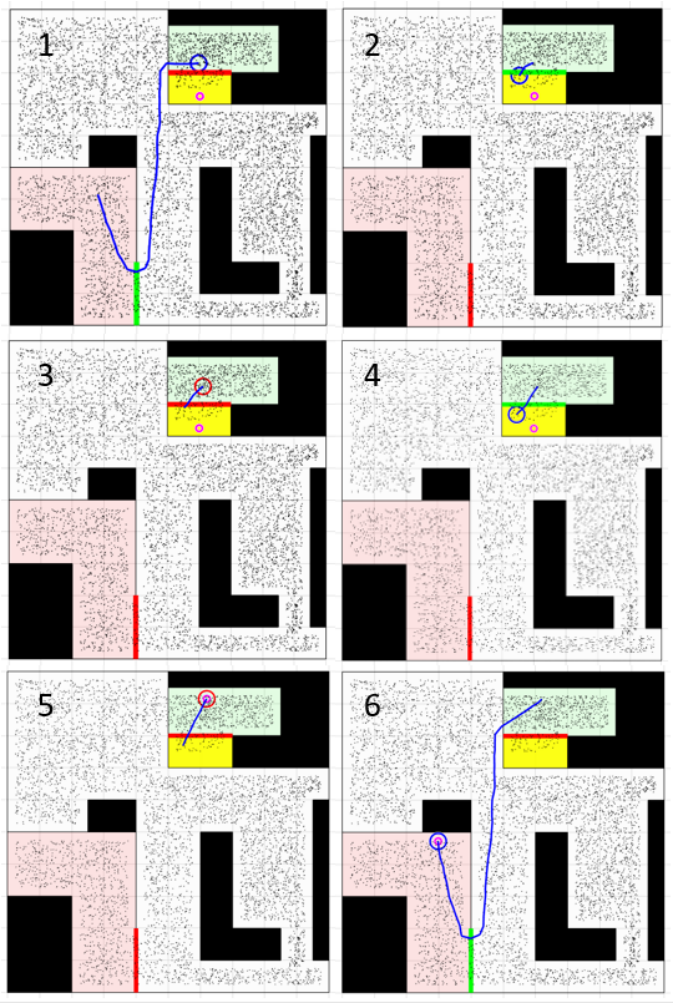

Fig. 7. Experiment to test the robot's error recovery behavior. The doors are red when closed, and green when open. 1. The robot opens the bedroom door and goes to the kitchen. 2. The robot opens the refrigerator and goes near it to access it. 3. The robot grabs the beer from the fridge and goes back to the kitchen, closing the refrigerator door. However, it fails to catch the beer. 4. The robot realizes this and goes back to the refrigerator. 5. The robot grabs the beer successfully this time. 6. The robot opens the bedroom door again and goes back to the bedroom.

V. Conclusion & Limitations

In this work, a task planner was designed with two hierarchical levels: a high-level semantic formulation of a LTL-synthesized controller, and a lower-level interpreter that executes these semantically-labeled states and action primitives. We justified this formulation by pointing out existing limitations of LTL-synthesized controllers that discretize the continuous space dynamics of an agent. This framework is formulated and tested by synthesizing a semantic controller for household environments, subjects to some environment constraints such as doors and objects it must manipulate. Through the simulation we verify that the framework leads to more optimal and tractable results, which are able to take dynamic environments and also recover from errors.

Below, we list some limitations and potential future works related to the framework.

A. Auto-Generating Progression LTL Specifications

We verified that the controller was able to plan the task with environment-awareness, by interpreting LTL safety specifications as a declarative knowledge database on how the environment works. However, one can argue that this did not achieve end-to-end task planning since we still have to supply the progression specification. For this work, how to translate "get the beer from the fridge" to our progression specification completely relied on heuristics. We mention some possible future directions that one might take to convert such natural-language commands into formal-logic specifications.

In its current state, PDDL is better fit for purely parsing these commands into steps compared to LTL-synthesized controllers, and one could conceive using PDDL to generate a sequence of specifications, given our action-primitives. Another possibility is to attempt to convert natural-language directly into formal logic, as described in [36]. Otherwise, one can directly attempt to parse natural-language and analyze it, as described in [37].

B. Characterizing Temporal Reactions

One disadvantage of our framework is that compared to synthesized controllers that discretize continuous-space directly, we must be very careful about how and what to communicate between the higher-level semantic controller, and the lower-level interpreter. We have justified our abstraction function that maps the continuous-space coordinates to semantic elements. However, in linear temporal logic, temporal space is also highly abstracted without any quantification.

In this sense, we can consider issues of timing issues such as synchronization and termination. For instance, how does the high-level controller know when the lower-level interpreter has finished its commanded action? There are two-solutions that could be potentially explored. The first solution is to receive some feedback from the lower-level interpreter that it has completed its action, which would tie the two layers temporally. For instance, a PRM motion planner can issue a termination cue when it has finished moving. Since we assume that the lower-level interpreter has executed correctly, this would be the open-loop solution.

Another solution would be to continuously query sensor variables and always verify that the lower-level interpreter indeed did carry out what it is supposed to do, constructing a closed-loop solution. The advantage of closed-loop solutions are that it can continuously sense errors and initiate the recovery behavior. One can quickly verify that the example in Sec.IV.D was possible only because we had a sensor variable to confirm that the robot indeed has the beer. However, this solution leads to some interesting other questions - how fast do we query these sensor variables, and when?

For instance, let us assume that we set the sensor query to run at a fixed interval of low frequency, and the synthesized controller reacted to this query at the same frequency. Then this will not only check for error recovery, but will generate some unnecessary behavior as well. An illustrative example is given by the following simple FTS in Fig. 8.

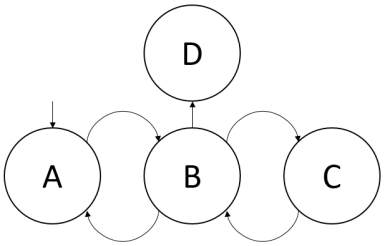

Fig. 8. A Simple FTS

Let us say that the agent has two specifications:

1) If the sensor is on and the agent is at *A*, go to *C*
2) If the sensor is on and the agent is at *B*, go to *D*

In this case, how fast we query the sensor makes a difference on where the agent ends up. This is problematic for our framework because if the PRM is taking the robot from *A* to *C*, but the robot has to pass through *B*, it will take it to *D* instead. Thus we must look for a right time to get sensor input and make the transition.

C. Semantics of Synthesized Controllers: Variables vs. States

Finally, we discuss some ambiguities in the usage of synthesized controllers. If we consider the door example above, multiple ways exist that we can model the door in a controller-synthesis problem. In our work, the doors are included as a system variable because the robot can manipulate whether or not they are closed or open. However, we can also place them in environment variables and define the action "open" instead. The final option is to augment the semantic states [17] according to whether or not the doors are open. In implementation, they may all result in equivalent synthesized automatas. However, it would be interesting to argue what makes more sense in a controller-design perspective.

ACKNOWLEDGMENTS

I would like to thank Dr. Petter Nilsson, who familiarized me with Linear Temporal Logic and the TuLiP toolbox. I would also like to thank Prof. Joel Burdick for very helpful advice on how to direct the work, and for three wonderful terms of learning various problems in robotics.

REFERENCES

[1] R.E. Fikes, N.J. Nilsson, "STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving", Artificial Intelligence Vol. 2, pp. 189-208, Winter, 1971

[2] E. Pednault, "Formulating Multi-Agent Dynamic-World Problems in the Classical Planning Framework", Reasoning about Actions and Plans, pp.47-82, 1987

[3] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, D. Wilkins, "PDDL - The Planning Domain Definition Language", Technical Report CVC TR98003/DCSTR1165, Yale Center for Computational Vision and Control, 1998

[4] M. Ghallab, D.S. Nau, P. Traverso, "Automated Planning: Theory and Practice", 2004

[5] C. Dornhege, "Task-Planning for High-Level Robot Control", Ph.D Thesis, University of Breiburg, November 2015

[6] C. Dornhege, A. Hertle, "Integrated Symbolic Planning in the Tidyup-Robot Project", AAAI Spring Symposium, 2013

[7] J. Frank, K. Golden, A. Jonsson, "The Loyal Opposition Comments on Plan Domain Description Languages", NASA Ames Research Center, 2003

[8] P. Gastin, D. Oddoux, "Fast LTL to Buchi Automata Translation", Proceedings of the 13th International Conference on Computer Aided Verification, 2001

[9] G.J. Holzmann, "The Model Checker SPIN", IEEE Transactions on Software Engineering - Special Issue on Formal Methods in Software Practice", Vol. 23, Issue 5, May 1997

[10] A. Cimmati, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, A. Taccella, "NuSMV2: An OpenSource Tool for Symbolic Model Checking", Proceedings of Computer Aided Verification, 2002

[11] H. Kress-Gazit, G.E. Fainekos, G.J. Pappas, "Temporal Logic-based Reactive Mission and Motion Planning", IEEE Transactions of Robotics, Vol. 25, Issue 6, Dec. 2009

[12] A. Kumar, "Linear Temporal Logic-Based Motion Planning", Master Thesis Presentation, Indian Institute of Technology, Allahabad

[13] M. Klotzer, C. Belta, "LTL Planning for Groups of Robots", IEEE International Conference on Networking, Sensing, and Control", 2006

[14] I. Filippidis, S. Dathathri, S.C. Livingston, N. Ozay, R.M. Murray, "Control Design for Hybrid Systems with TuLiP: The Temporal Logic Planning Toolbox", IEEE Conference on Control Applications (CCA), 2016

[15] R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, Y. Sa'ar, "Synthesis of Reactive (1) Designs", Journal of Computer and System Sciences", Vol. 78, Issue 3, pp.911-938, May 2012

[16] P. Tabuada, G.J. Pappas, "Linear Time Logic Control of Discrete Time Linear Systems", TAC, Vol. 51, No. 12, pp. 1862-1877, 2006

[17] P. Nilsson, N. Ozay, J. Liu, "Augmented Finite Transition Systems as Abstractions for Control Synthesis", Discrete Event Dynamic Systems 27.2, pp.301-340, 2017

[18] P. Nilsson, "Correct-by-Construction Control Synthesis for High-Dimensional Systems", Ph.D Thesis, University of Michigan, 2017

[19] E. Galceran, M. Carreras, "A Survey on Coverage Path Planning for Robotics", University of Girona, Underwater Robotics Research Center (CIRS), August 2013

[20] C. Stachniss, G. Grisetti, W. Burgard, "Information Gain-Based Exploration Using Rao-Blackwellized Particle Filters", Robotics: Science and Systems, 2005

[21] L.E. Kavraki, P. Svestka, J.C. Latombe, M.H. Overmars, "Probablistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces", IEEE Transactions on Robotics and Automation, Vol. 12, No. 4, August 1996

[22] V. Raman, C. Lignos, C. Finucane, K.C.T. Lee, M. Marcus, H. Kress-Gazit, "Sorry Dave, I'm Afraid I can't Do That: Explaining Unachievable Robot Tasks Using Natural Language", Robotics: Science and Systems, 2013

[23] E.L. Gunter, "From Natural Language to Linear Temporal Logic: Difficulties of Capturing Natural Language Specifications in Formal Languages for Automatic Analysis", Monterey Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation, Sep 2003

[24] C. Galindo, J. Fernandez-Madrigal, J. Gonzalez, A. Saffiotti, "Robot Task Planning Using Semantic Maps", IEEE/RSJ International Conference on Intelligent Robots and Systems, 2005

[25] J.R. Ruiz-Sarmiento, C. Galindo, J. Gonzalez-Jimenez, "Robot@Home, a Robotic Dataset for Semantic Mapping of Home Environments", International Journal of Robotics Research (IJRR), 2017

[26] N. Sunderhauf, F. Dayoub, S McMahon, B. Talbot, "Place Categorization and Semantic Mapping on a Mobile Robot, IEEE International Conference on Robotics and Automation (ICRA), May 2016

[27] D. Pangeric, B. Pitzer, M. Tenorth, M. Beetz, "Semantic Object Maps for Robotic Housework - Representation, Acquisition, and Use", IROS 2012

[28] R. Brunelli, "Template Matching Techniques in Computer Vision: Theory and Practice", Wiley Publications, April 2009

[29] A. Saxena, J. Driemeyer, A. Y. Ng, "Robotic Grasping of Novel Objects using Vision", IJRR Vol. 27, Issue 2, 2008

[30] P. Hebert, N. Hudson, J. Ma, T. Howard, T. Fuchs, M. Bajracharya, J. W. Burdick, "Combined Shape, Appearance, and Silhouette for Simultaneous Manipulator and Object Tracking", ICRA, May 2012

[31] A.T. Miller, P.K. Allen, "Graspit! A Versatile Simulator for Robotic Grasping", IEEE Robotics & Automation Magazine, Vol. 11, Issue 4, Dec 2004

[32] Y. Karayiannidis, C. Smith, F. E. Vina, P. Ogren, D. Kragic, "Open Sesame! Adaptive Force/Velocity Control for Opening Unknown Doors", IROS 2012

[33] S. Chinchali, S.C. Livingston, U. Topcu, J. W. Burdick, R. M. Murray, "Towards Formal Synthesis of Reactive Controllers for Dexterous Robotic Manipulation", ICRA, May 2012

[34] E. Plaku, "Path Planning with Probabilistic Roadmaps and co-safe Linear Temporal Logic", IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012

[35] S.C. Livingston, R.M. Murray, J.W. Burdick, "Backtracking Temporal Logic Synthesis for Uncertain Environments", ICRA, 2012

[36] R.J. Kate, Y.W. Wong, R. J. Mooney, "Learning to Transform Natural to Formal Languages", AAAI 2005

[37] H. Khayrallah, S. Trott, J. Feldman, "Natural Language for Human Robot Interaction", International Conference on Human-Robot Interaction (HRI), 2015