future internet

MDPI

Article

Semantic Task Planning for Service Robots in Open Worlds

Guowei Cui, Wei Shuai and Xiaoping Chen *

School of Computer Science and Technology, University of Science and Technology of China, Hefei 230026, China; cuigw@mail.ustc.edu.cn (G.C.); swwsag@mail.ustc.edu.cn (W.S.)
* Correspondence: xpchen@ustc.edu.cn

Abstract: This paper presents a planning system based on semantic reasoning for a general-purpose service robot, which is aimed at behaving more intelligently in domains that contain incomplete information, under-specified goals, and dynamic changes. First, Two kinds of data are generated by Natural Language Processing module from the speech: (i) action frames and their relationships; (ii) the modifier used to indicate some property or characteristic of a variable in the action frame. Next, the task's goals are generated from these action frames and modifiers. These goals are represented as AI symbols, combining world state and domain knowledge, which are used to generate plans by an Answer Set Programming solver. Finally, the plan's actions are executed one by one, and continuous sensing grounds useful information, which makes the robot use contingent knowledge to adapt to dynamic changes and faults. For each action in the plan, the planner gets its preconditions and effects from domain knowledge, so during the execution of the task, the environmental changes, especially those conflict with the actions, not only the action being performed but also the subsequent actions, can be detected and handled as early as possible. A series of case studies are used to evaluate the system and verify its ability to acquire knowledge through dialogue with users, solve problems with the acquired causal knowledge, and plan for complex tasks autonomously in the open world.

Keywords: general purpose service robot; task planning; open world

Citation: Cui, G.; Shuai, W.; Chen, X. Semantic Task Planning for Service Robots in Open Worlds. *Future Internet* **2021**, *13*, 49. https://doi.org/10.3390/fi13020049

Academic Editor: Jacopo Soldani
Received: 17 January 2021
Accepted: 12 February 2021
Published: 17 February 2021

1. Introduction

Research on service robots has received increasing attention in recent years [1–3]. In most scenarios, like nursing homes and offices, humans hope that robots can help them do many tasks, which include taking orders and serving drinks, welcoming and guiding guests, or just cleaning up. To achieve that goal, a service robot requires human-like information processing and the underlying mechanisms for dealing with the real world, especially the ability to communicate with humans, acquire the knowledge to complete tasks and adapt to the dynamic environment.

For most users, speech is preferable to other communication methods with a general-purpose service robot in most scenarios. Speech is also the primary communication method in General-Purpose Service Robot (GPSR) test defined by RoboCup@Home https://athome.robocup.org (accessed on 12 February 2021). The user verbally assigns a complicated task to the robot that may require a set of skills. The robot needs to perform the task, report any problems, adapt to unexpected changes, and find alternative solutions with brief knowledge about the domain. Automated planning, which makes decisions about how to act in the world, requires symbolic representations of the robot's environment and the actions the robot can perform, has been widely used for task planning and control in many service robot applications. In the open world, there are two main challenges for task planning: 1) the robot's perception of the world is often incomplete, a command may refer to an object that is not in its knowledge base, lack of information will fail to generate a plan; 2) changes in the dynamic environment may not be expected by the robot, which will cause the planned action to fail.

This paper addresses these issues by developing a semantic task planning system, which combines natural language understanding, task-oriented knowledge acquisition, and semantic-based automated task planning.

Research in knowledge representation (KR) and logical reasoning has provided sophisticated algorithms [4–6], which have been used on service robots for supporting task planning and execution. Stenmark et al. [7] attempt to combine descriptions of robot tasks using natural language together with their realizations using robot hardware to representing knowledge about industrial processes. The Kejia robot [8] represents domain knowledge learned through natural language processing and leverages a symbolic planner for problem-solving and planning to provide high-level functions [9,10]. The system has been extended to acquire task-oriented knowledge by interacting with the user and sensing the environment [11]. By interacting and sensing and grounding useful sensing information, the robot can work with incomplete information and unexpected changes. Abukhalil et al. [12] present a high-level configuration and task assignment software package that distributes algorithms on a swarm of robots, which allows them to operate in a swarm fashion. Savage et al. [3] use a conceptual-dependency [13] interpreter extracts semantic role structures from the input sentence and planning with the open-source expert system CLIPS [14]. Puigbo et al. [15] adopt Soar cognitive architecture [16] to support understanding and executing human-specified commands. In [17], the robot's knowledge for safety is enhanced by using data coming from sensors with knowledge from reasoning. Respecting safety requirements, [18] aims to redesign existing industrial robotic cells for executing a number of collaborative actions based on a knowledge base. In this work, the robot maintains a certain distance from obstacles or people to ensure safety. Moreover, voice is also used to remind users to pay attention to safety. Similar to these works, our system combines a KR system and an ASP planner for high-level planning.

Planning approaches that work in open-world scenarios often need to find a way to close the world. Using Open World Quantified Goal, [19,20] can bias the planner's view of the search space towards finding plans that achieve additional reward in an open world. To address incomplete information in the open world, methods of planning with HRI and sensing actions are developed. Petric et al. utilize a collection of databases, each representing a different kind of knowledge [21]. Some methods [15,22] collects information from user during natural language processing (NLP). representing HRI actions using planning actions [23]. Some works collects task-oriented information by combining HRI with planning [11,24–26]. Some work focus on using open source knowledge to handle the incomplete information [10,27]. Planning with sensing actions has been investigated under different semantics and specific planning algorithms [21,28,29]. Mininger and Laird [30] use a Soar-based interactive task-learning system to learn strategies to handle references to unseen objects. The approach defines a "find" subtask with a special postcondition so the system can succeed in planning for tasks requiring direct interaction with unseen objects. In [31], sensor information is used to update an ontology that is queried in each planning loop to populate a PDDL (Planning Domain Definition Language) [32] problem file. Petrick et al. [33] extensions to the knowledge-level PKS (Planning with Knowledge and Sensing) planner to improve the applicability of robot planning involving incomplete knowledge. Lim et al. [34] present an ontology-based unified robot knowledge framework that integrates low-level data with high-level knowledge to enables reasoning to be performed when partial information is available. Hanheide et al. [2] extends an active visual object search method [35] to explain failures by planning over explicitly modeled additional action effects and assumptive actions. Jiang et al. [1] provide an open-world task planning approach for service robots by forming hypotheses implied by commands of operators. Similar to [1,2], assumptions and grounding actions are adopted in this paper, but only when there are not enough objects to meet the user's instructions, the corresponding assumptions will be generated.

Another research area related to this work focuses on planning with dynamic change and uncertainty. Zhang et al. use commonsense reasoning to dynamically construct

POMDPs (Partially Observable Markov Decision Processes) for adaptive robot planning [36,37]. Using theory of intensions [38], which is based on scenario division, [39] generates plan with loops to accommodate unknown information at planning time. In [40], a semantic knowledge base is proposed to derive and check implicit information about the effects of actions during plan generation. In this paper, a classic "plan-execute-monitor-replan" is used to handle unexpected changes in the open world.

For the first problem (incomplete information), the idea is "close" the world, which means each object involved in the command must be known in the knowledge base. *Assumption* and *grounding* operations are involved to handle this. First, the natural language processing module will generate two outputs: 1) action frames and their relationships; 2) modifier used to indicate some property or characteristic of a variable in the action frame. Next, the action frame and modifiers are used to generate *goals* of the task. For the object in the command that is not in the knowledge base, an assumption will be added to the knowledge base. A grounding operation will finally check this assumption is true or not when the operation is executed.

For the second problem (dynamic environment), the environment is dynamic, and robots must be able to start from incomplete information, gather useful information, and achieve the goals. In order to response the dynamic environment, *continuous perception* and *conflict detection* are adopted. We formalize continuous sensing in a formal representation, which is transformed into Answer Set Programming (ASP) [41] to generate plans by an ASP solver [42], and the robot perform plans using the classical "plan-execute-monitor-replan" loop. The monitor checks if the change conflicts with actions in the plan, the action being performed and the subsequent actions, so the conflict can be detected and handled as early as possible. Our method features: 1) a method of confirming task type, extracting the roles of the task and the roles' constrained information; 2) assumption and grounding methodology to "close" the open-world, it is only introduced when there are not enough instances, which can benefit from the use of existing knowledge; 3) continuous sensing and conflict detection mechanism for all actions not performed that captures dynamical changes in the environments and triggers special processing as soon as possible. Similar to our work is [1] and [11], compared with them, our algorithm can find potential conflicts earlier and better balance assumptions and existing knowledge to reduce the cost of the task.

This paper is organized as follows—we describe the overview of the system in Section 2. Next, we describe the knowledge representation and domain formulation in Section 4 and natural language understanding in Section 3. Then in Section 5, where the implemented "plan-execute-monitor-replan" techniques are described. Experimental results and evaluations are presented in Sections 6.

2. Framework Overview

An architecture is briefly introduced in this section. A central problem in robotic architectures is how to design a system to acquire a stable representation of the world suitable for planning and reasoning.

In general, a service robot should be able to perform basic functions:

- Self-localization, autonomous mapping, and navigation.
- Object detection, recognition, picking, and placement.
- People detection, recognition, and tracking.
- Speech recognition and Natural Language Understanding.

A long-standing challenge for robotics is how to act in the face of uncertain and incomplete information, and to handle task failure intelligently. To deal with these challenges, we propose a framework for service robots to behave intelligently in domains that contain incomplete information under specified goals and dynamic change. The framework is composed of five layers: Input, Knowledge Management, Planning, Execution, and Monitor.

This section will describe the most relevant modules categorized by layer. Figure 1 depicts the information interaction between the layers.

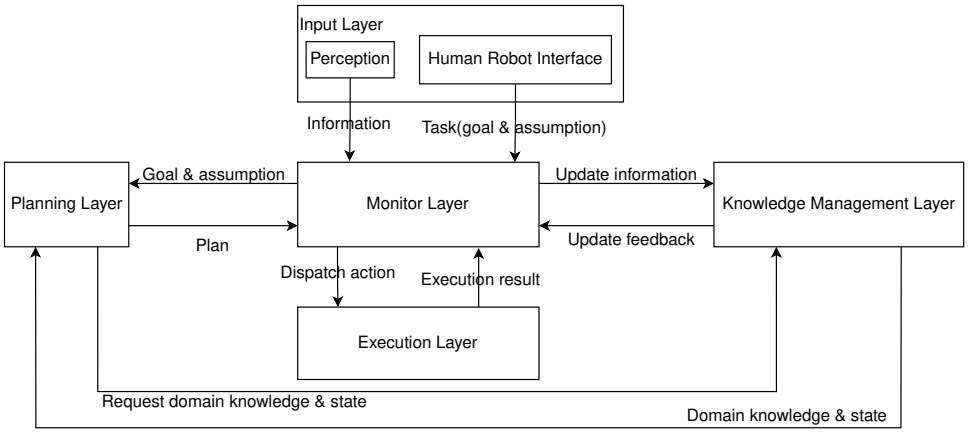

Figure 1. Framework overview.

2.1. Input Layer

This layer involves the modules that provide necessary facilities for sensing its environment and communication with other agents, mainly includes two modules: perception and human-robot interaction.

2.1.1. Perception

This module aims to sense the environment, which has several sub-modules.

- Self-localization and autonomous mapping.
- Object detection and recognition.
- People detection and recognition.

This module generates beliefs about the possible states of the environment. Beliefs are based on the symbolic representation of the sensory information from internal and external sensors. These beliefs are finally transferred to Knowledge Management via Monitor and used to update the state of the world.

2.1.2. Human Robot Interface

This module contains two parts: Speech Recognition and Natural Language Understanding. HRI provides the interface for communication between users and the robot.

Speech Recognition uses the Speech Application Programming Interface (SAPI) developed by iFlytek The speech will be processed in the NLP module, as explained in Section 3. Based on the user's instructions, NLP generates assumptions about the possible states of the environment and goals that represent the user's intents. These assumptions are based on the symbolic representation of the information from users. These assumptions and goals are transmitted to the Planner via Monitor to trigger a plan.

2.2. Knowledge Management Layer

This layer involves all modules that store and access the robot's knowledge. Such knowledge, which is symbolic, includes structuring the informational state of the world, goals, and domain knowledge.

For high-level reasoning, a rule-based system is used. The facts and rules are written in ASP [41] format and represent the robot's knowledge as explained in detail in Section 4.

2.3. Planning Layer

This layer is responsible for generating plans at a high level of abstraction and performing global reasoning.

Assumptions transferred are added into init states. Together with domain knowledge and goals, they are used to trigger the Action Planner, which will generate a sequence of actions to achieve the desired goals.

2.4. Monitor Layer

This layer dispatches the generated plan to the execution layer, monitors the execution and open-world changes. By "plan-execute-monitor-replan", if something unexpected happens while executing a plan, the Monitor will interrupt the execution and trigger a new plan.

2.5. Execution Layer

This layer controls the robot to execute the generated plans. Each step of the plan is an atomic function that solves a specific problem. These functions should be simple, reusable, and easy to implement with a state machine.

3. Natural Language Understanding

This section describes the NLP technology employed by this work. The NLP is in charge of translating the speech from users into a symbolic representation used by the action planner.

For each sentence, NLP finds the main event. After finding the main event in such a sentence, it must determine the roles played by the sentence elements and the conditions under which the event takes place. The verb in a sentence usually is used to find a structure of the event composed of participants, objects, actions, and relationships between event elements. These relations can be temporal or spatial. For example, in "*Robot, find Mary, and bring her an apple*", *Robot* is the actor, *her* is the recipient, *apple* is the object, and *an* represents the number of the *apple*. According to the context, the NLP module should figure out who *her* is referring to.

For each input sentence, the NLP module works in three steps: (1) Parsing, in which Stanford Parser [43] parses the sentences and outputs grammatical relations as typed dependencies; (2) Semantic analysis, in which typed dependencies are used to generate action frames; (3) Goals generation, in which action frames are translated into the logic predicates that can be recognized by an ASP solver.

3.1. Parsing

The NLP module's input from the human-robot dialog is a string of words that is regarded as a sentence. This sentence is parsed by the Stanford parser, which works out the grammatical structure of sentences, can offer two kinds of information: a grammar tree with UPenn tagging style, and a set of typed dependencies with Universal Dependencies style or Stanford Dependencies style. These typed dependencies are otherwise known as grammatical relations.

In our system, we use universal dependencies v1 [44]. The idea of universal dependencies is to propose universal grammatical relations that can be used with relative fidelity to capture any dependency relation between words in any language. There are 40 universal relations; here's a brief introduction to part relationships that play an essential role in semantic analysis.

The core dependencies play the most important role in getting semantic elements of an action or event. We mainly consider three core dependencies:

- *nsubj*: nominal subject. The governor of this relation is a verb in most cases, and it may be headed by a noun, or it may be a pronoun or relative pronoun.
- *dobj*: direct object. Typically, the direct object of a verb is the noun phrase that denotes the entity acted upon or which changes state or motion.
- *iobj*: indirect object. In many cases, the indirect object of a verb is the recipient of ditransitive verbs of exchange.

Modifier word is also an important type of dependency, we consider *amod*, *nummod*, *det*, *neg*, *nmod* in our system.

- *amod*: adjectival modifier. An adjectival modifier of a noun is an adjectival phrase that serves to modify the meaning of the noun.
- *nummod*: numeric modifier. A numeric modifier of a noun is any number phrase that serves to modify the meaning of the noun with a quantity.
- *det*: determiner. The relation determiner (det) holds between a nominal head and its determiner. Determiners are words that modify nouns or noun phrases and express the reference of the noun phrase in context. That is, a determiner may indicate whether the noun is referring to a definite(words like many, few, several) or indefinite(words like much, little) element of a class, to an element belonging(words like your, his, its, our) to a specified person or thing, to a particular number or quantity(like words any, all), and so forth.
- *neg*: negation modifier. The negation modifier is the relation between a negation word and the word it modifies.
- *nmod*: nominal modifier. It is a noun (or noun phrase) functioning as a non-core (oblique) argument or adjunct. This means that it functionally corresponds to an adverbial when it attaches to a verb, adjective, or adverb. However, when attaching to a noun, it corresponds to an attribute or genitive complement (the terms are less standardized here).

3.2. Semantic Analysis

To get the semantic representation, which is a set of semantic elements, typed dependencies are required, and sometimes the syntactic categories of words and phrases are required, too. These typed dependencies are used to generate action frames and modifiers.

3.2.1. Action Frame

A semantic role refers to a noun phrase that fulfills a specific purpose for the action or state that describes the main verb of a statement. The complete description of the event can be modeled as a function with parameters that correspond to semantic roles in an event that describes the verb, such as actor, object, start and destination place. An action frame is generated from typed dependencies. The frame contains five elements: *action(Actor, Action, Object, Source, Goal)*.

- *Actor*: The entity that performs the *Action*, the *Actor* is an agent that usually is a person or a robot.
- *Action*: Performed by the *Actor*, done to an *Object*. Each action primitive represents several verbs with a similar meaning. For instance, give, bring, and take have the same representation(the transfer of an object from one location to another).
- *Object*: The entity the *Action* is performed on. It should be noted that the *Object* can also be a person or robot. For instance, from sentence *"bring James to the office"*, an action frame *action(NIL, bring, James, NIL, office)* is generated, where *NIL* represents an empty slot that needs to be filled according to the content and domain knowledge.
- *Source*: The initial location of *Object* when the *Action* starts.
- *Goal*: The final location of *Object* when the *Action* stops.

Usually, from the core dependencies, such as *nsubj* and *dobj*, an action frame's *Actor*, *Action*, *Object* can be identified. The slots *Source* and *Goal*, both of them are always associated with prepositional phrase *PP* and dependency *nmod*.

For instance, with "take this book from the table to the bookshelf" as an input, Stanford Parser outputs the following result.

Parser tree:

```
(ROOT
  (S
    (VP (VB take)
      (NP (DT this) (NN book))
```

```
      (PP (IN from)
        (NP
          (NP (DT the) (NN table))
          (PP (TO to)
            (NP (DT the)
              (NN bookshelf))))))))
```

Typed dependencies:

```
root(ROOT-0, take-1)
det(book-3, this-2)
dobj(take-1, book-3)
case(table-6, from-4)
det(table-6, the-5)
nmod:from(take-1, table-6)
case(bookshelf-9, to-7)
det(bookshelf-9, the-8)
nmod:to(table-6, bookshelf-9)
```

The tag *VB* is used to identify the *Action*, and the dependency *dobj* is the core relation used to determine the *Object*. From *nmod:from* and *nmod:to*, the *Source* and *Goal* location of the action frame are extracted. No *Actors* are found from the parsing result, obviously, the *Actor* can only be identified by content, and that will be the person or robot who talks to the person who said this sentence.

3.2.2. Modifier

A modifier is a word, phrase or clause that modifies other elements of a sentence. Nouns, adjectives, adjective clauses and participles can be used as modifiers of nouns or pronouns; A quantifier word is used in conjunction with a noun representing a countable or measurable object with a number, often used to indicate a category. To be more intuitive, there are some examples in Table 1.

Table 1. Some modifier examples.

Phase	POS	Dependency	Description
big cup	JJ big NN cup	amod(cup, big)	The size of the cup, which is big siz
two apples	CD two NNS apples	nummod(apples, two)	The number of the apples.
no apples	DT no NNS apples	neg(apples, no)	Zero.
my book	PRP$ my NN book	nmod:poss(book, my)	Possession, I own this book.

An *Modifier*, indicates some attribution of an object with some value. For example, dependency *nummod(apples, two)* is represented as *number(apple, 2)*. The word *number* means the attribute, the number *2* is the value, and *apple* is the object. These modifiers provide conditionality for the elements in the action frame.

3.2.3. Pronoun

Pronouns are often used, so an important task is to figure out the noun substituted by the pronoun. It is necessary, or the robot has to ask who or what the pronoun refers to. Our system performs a primitive type of deduction according to the closest matching principle. In every sentence, the noun, such as actor, object or location recognized, is saved. When a pronoun appears in a later part or a new sentence, the saved nouns' closest matching is

used to replace the pronoun. Matching is based on the pronoun's possible meaning and the restrictions in the sentence, such as the action acting on the pronoun. For instance, when the user says *"grasp a cup, go to the living room, and give it to Mary"*, *it* will be recognized as *cup* rather than *living room* due to the room is immovable.

3.3. Goals Generation

In our system, the ultimate goal of NLP is to generate goals that should be achieved by executing the plan solved by the solver according to the current world state. Table 2 lists some actions and corresponding goals.

Table 2. Action frame and corresponding goal.

Action Frame	Goal	Assumption
$action(A, get, O, F, T)$	$in(O, A)$	$in(O, F)$
$action(A, give, O, F, T)$	$in(O, T)$	$\varnothing$
$action(A, move, O, F, T)$	$in(A, T)$	$in(A, F)$
$action(A, find, O, F, T)$	$in(O, F)$	$in(O, F)$

The predicate in can be transformed into the required predicate according to the category of its parameters. For example, $in(O, A)$ can converted into $isHeld(O, A)$, which indicates O is held by A, when O is an graspable object and A is a person or robot. The assumption is information derived from the user's instructions but has not yet been confirmed by the robot. Each assumption needs to be identified by the $find$ operator.

In the user's instructions, the object or place may not be unique, and this generic object or place constraint needs to be added to the goal representation. Besides, the objects involved in the command may have additional constraints, which are usually represented by modifiers and need to be added to the target representation. For instance, *"give Mary two apples"*, its corresponding action frame is $action(robot, give, apple, None, Mary)$ and corresponding modifier is $number(apple, 2)$, its goal in a clingo program is shown below. (The encodings follow the style of the examples in the CLINGO guide: https://github.com/potassco/guide (accessed on 12 February 2021).)

$$2\,\{\,in(X, mary) : apple(X)\,\}\,2.$$

4. Knowledge Representation

Knowledge in this work represents type hierarchy, domain objects, states, and causal laws. The causal laws include the fluents of physical actions, HRI actions, and sensing actions. We particularly focus on three kinds of knowledge. (i) **Domain knowledge**, including causal laws that formalize the transition system and knowledge of entities(such as objects, furniture, rooms, people, etc.). (ii) **Control knowledge**. When the robot faces a dining table and about to pick up *Pepsi*, it will measure the distance of the object to determine if it should move closer, adjust its gripper, or fetch. (iii) **Contingent knowledge**. Throughout performing the task, the robot should continuously observe the environment, gather useful information, enrich its knowledge, and adapt to the change.

Answer Set Programming (ASP) [41] is adopted as knowledge representation and reasoning tool. It is based on the stable model (answer set) semantics of logic programming. When ASP is used for planning, an action model is not divided into precondition and effect like PDDL [32]. Our system needs to check preconditions and effects before or after performing each step in a plan, so we code the action model according to the precondition and effect, then convert it into an ASP program.

4.1. Domain Knowledge

Domain knowledge consists of two kinds of knowledge:

- The information(type, position, etc.) related to objects(humans, objects, locations, rooms, and robots).
- Causal laws that formalize the transition system.

Types. We use a rigid (time-independent) fluent in denoting type and object membership. For instance, we denote *pepsi1* is a member of type *pepsi* by pepsi(pepsi1). Type hierarchy and relations are formalized using static laws such as *obj(X)* if *pepsi(X)*. There are five top types in our system, and each type has some sub-types. As Figure 2 shows, the leaf nodes are memberships, and internal nodes are types of entities. The parent node of *pspsi1* is pepsi, and *pepsi*'s parent node is *drink*. Different types of entities have various properties. Each of them corresponds to a predicate, corresponding, and different information. These properties are used for object filtering.

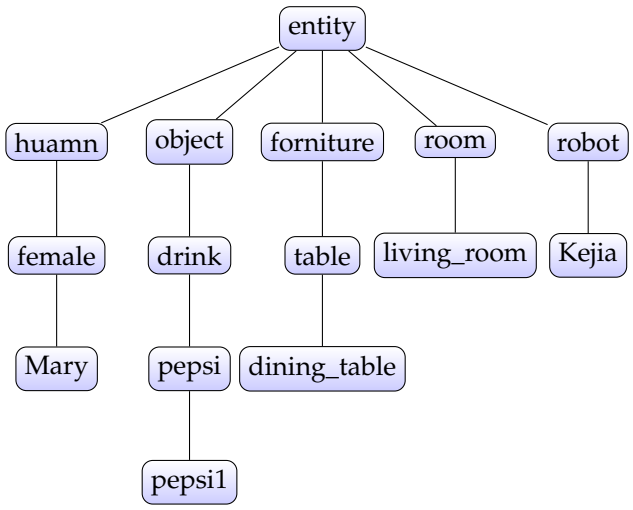

Figure 2. Illustrate the type hierarchy. There are five top types in our system: robot, human, furniture, room, and object.

States. Non-rigid fluents, each of them means a particular situation that the object is in: the location of objects, the position relation between objects, the gripper is empty or not, the object is on a table or not, a container or door is open or closed, the robot's position, and so forth.

Static laws. The transformation of fluents is independent of actions and time. Static rules can describe category relations, inherent properties, and so forth. For instance, *"pepsi is a drink"*, from *pepsi(pspsi1)*, *drink(pepsi1)* can be generated by a static causal law:

$$drink(A) :- pepsi(A).$$

Dynamic laws. The rules of describing the changing of fluents over time or actions play a core role in action planning. These rules fall into two categories:

1. The changing of states over time. In our system, A state will be transferred to the next moment without interference from external forces, as shown by Equation 1. In Equation 1, X means a state and $\_T$ means time (step).
2. The transition under an operation.

$$holds(X, \_T) :- holds(X, \_T - 1), not \neg holds(X, \_T), \_T <= n. \quad (1)$$

The transition with action has the following basic components:

1. *Action*, an operation that the robot can perform to interact with the environment.
2. *Preconditions*, a set of states, that need to be met before or during an action.
3. *Effects*, a set of states that can be achieved at the end of an action.

The following is the actions in our system.

1. **moveTo(L,R)**: The robot navigates to the place L in the room R.
2. **moveIn(R2,R1,D)**: The robot navigates to the room $R2$ through the door D from the room $R1$.
3. **pickup(O,L,G)**: The robot pick up the object O from the location L that it will carry in its actuator G.
4. **putdown(O,L,G)**: The robot releases the object O that it carries in its actuator G in the specified place L.
5. **findPerson(H,R)**: The robot searches the target person H at the room R.
6. **findObj(O,L)**: The robot activates object recognition to find a target object O from the location L.
7. **handOver(O,H,G)**: The robot hands the person H the object O in its hand G.

Table 3. Action, its preconditions and effects in our system.

Action	Preconditions	Effects
$moveTo(L, R)$	$inRoom(robot, R)$ AND (NOT $isNear(robot, L)$) AND ($inRoom(L, R)$ OR $locInRoom(L, R)$)	$isNear(robot, L)$ AND ($\neg isNear(robot, L1)$ IF $isNear(robot, L1)$)
$moveIn(R2, R1, D)$	$inRoom(robot, R1)$ AND $door(D)$ AND $opened(D)$ AND $connected(D, R1, R2)$	$inRoom(robot, R2)$ AND $\neg inRoom(robot, R1)$ AND $isNear(robot, D)$ AND ($\neg isNear(robot, L)$ IF ($isNear(robot, L)$ AND $L \neq D$))
$pickup(O, L, G)$	$hand(G, robot)$ AND $empty(G)$ AND $isPlaced(O, L)$ AND $graspable(O)$ AND $isNear(robot, L)$	$\neg empty(G)$ AND $\neg isPlaced(O, L)$ AND $isHeld(O, G)$
$putdown(O, L, G)$	$isHeld(O, G)$ AND $isNear(robot, L)$ AND $canBePlaced(L)$	$\neg isHeld(O, G)$ AND $empty(G)$ AND $isPlaced(O, L)$
$findPerson(H, R)$	$assume(isLocated(H, R))$ AND $inRoom(robot, R)$ AND $human(H)$ AND (NOT $searched(R)$)	$\neg assume(isLocated(H, R))$ AND $inRoom(H, R)$ AND $searched(R)$ AND ($\neg isNear(robot, L)$ IF $isNear(robot, L)$)
$findObj(O, L)$	$assume(isPlaced(O, L))$ AND $isNear(robot, L)$ AND (NOT $scaned(L)$)	$\neg assume(isPlaced(O, L))$ AND $isPlaced(O, L)$ AND $scaned(L)$
$handOver(O, H, G)$	$hand(G, robot)$ AND $isHeld(O, G)$ AND $isNear(robot, H)$ AND $human(H)$	$\neg isHeld(O, G)$ AND $empty(G)$ AND $has(H, O)$

Table 3 lists their preconditions and effects. Equation 2 describes that when the goals are not reached, only one action can occur at the same moment. In Equation 2, A means an action and $\_T$ means time (step), $\_ngoal(\_T - 1)$ is a state means at step $\_T - 1$ that there are some goals are not reached.

$$1\,\{occurs(A, \_T)\ :\ \_allowed(A, \_T)\}\,1 :- \_ngoal(\_T - 1),\ \_T <= n. \tag{2}$$

In the system, the action models are transformed into ASP. Taking action *moveTo* as an example, its preconditions are transformed into Equation 3 and Equation 4. If either of the two is satisfied, the action is allowed.

$$\begin{aligned} \_allowed(moveTo(L,R),\_T) \text{ :- } & holds(inRoom(robot,R),\ \_T-1), \\ & holds(inRoom(L,R),\_T-1), \\ & not\ holds(isNear(robot,L),\_T-1). \end{aligned} \tag{3}$$

$$\begin{aligned} \_allowed(moveTo(L,R),\_T) \text{ :- } & holds(inRoom(robot,R),\ \_T-1), \\ & locInRoom(L,R), \\ & not\ holds(isNear(robot,L),\_T-1). \end{aligned} \tag{4}$$

Its effects are transformed into Equation 5 and Equation 6. The keyword **IF** describes an accompanying state transition. The conditions defined by **IF** is not necessary for the action, but when these conditions are met, the accompanying effects will be produced when the action occurs (as shown in Equation 6).

$$holds(isNear(robot,L),\_T) \text{ :- } occurs(moveTo(L,R),\_T). \tag{5}$$

$$\begin{aligned} \neg holds(isNear(robot,L1),\_T) \text{ :- } & occurs(moveTo(L,R),\_T), \\ & holds(isNear(robot,L1),\_T-1). \end{aligned} \tag{6}$$

4.2. Control Knowledge

Control knowledge is oriented to atomic operations in task planning in our system. Each atomic operation is implemented using one or more state machines. The goal of this knowledge is to accomplish tasks more efficiently. This knowledge is usually some control parameters applied to the state machine to complete the atomic operations.

For instance, when the robot performs a pick-up task, for example, grabbing a bottle of iced black tea, it will measure the distance between the object and the robot to determine if it should move closer, adjust its gripper, or fetch. The robot needs to figure out that the distance of objects may affect its manipulation strategy, figure out grabbing which part, and using which posture to grab to get a greater success rate. For a recognition task, like finding an apple, the sequence of searching places can be added to the domain language for task planning. However, the camera's angle adjustment and the distance between the object and the robot to complete the identification task more efficiently belong to the control knowledge, which is integrated into the visual control system.

4.3. Contingent Knowledge

In performing tasks, robots should continuously observe the environment, collect useful information, enrich knowledge, and adapt to changes. This is particularly important because objects in a domestic environment are continually changing, and the information provided by a human can be fuzzy or wrong. Therefore, robots must start from a local, incomplete, and unreliable domain representation to generate plans to collect more information to achieve their goals.

Continuous sensing is a mechanism that updates the current state when the perception module finds new information. It allows the robot to reduce the uncertainty of the domain while executing the actions. Therefore, the robot has more robust adaptability and robustness to changing fields and irresponsible actions.

The information discovered by sensing are encoded as **States** mentioned in 4.1. There are two types of knowledge effects/states the robot's actions can have: *belief* (I believe X because I saw it) and *assumption* (I'll assume X to be true) [2]. Assumptions are derived from the user's speech or historical observations, and the transformation from *assumption* to *belief* is achieved through continuous sensing. In planning, realizer actions, such as *findPerson* and *findObj*, are used to complete this transformation.

5. Planning, Execution and Monitor

The main control loop for plan generation and execution follows the traditional *planning-execution-monitor-replan* loop. Symbol grounding for sensing actions is handled the same way as in continuous observation. The execution result is compared with the expected state to determine if a re-plan is needed.

5.1. Planning

A robot task planning problem is defined by the tuple (S_0, G, D), where S_0 is the initial state, G is the goal condition, D is the domain knowledge desired in Section 4.1, which contains action models and other rules. A plan Π is a plan consisting of a set of action sequences $\Pi = \langle a_1, ..., a_n \rangle$ that start from S_0 to satisfy G.

5.1.1. Goal Condition

The goal G for planning is generated by parsing a spoken command. How to generate goal condition from a command is described in Section 3.

5.1.2. Initial State

The initial state S_0 for planning is generated as follows: (i) fluents that belong to definite world state are initialized based on robot's sensor inputs, aka *beliefs*; (ii) fluents that generated by NLP module through talking to people or experiences, aka *assumptions*; (iii) fluents that belong to belief state are initialized as negated literals, denoting that the robot does not know anything about them. All these fluents are stored in the database. After the task's goal is generated, some of the fluents are extracted from the database to form the initial state. Only some of them are extracted to speed up the planning. The extracted parts include: properties and states of all rooms and furniture, properties and states of people and objects involved in the goal condition.

In the open world, robots are not omniscient about the world's state. So one of the problems is that the objects or people in the task are unknown. For instance, in the task "*bring me a pepsi*", there are no instances of *pepsi*. At this point, from the initial state, the goal is not reachable in the planning. We use *assumptions* to solve the challenge, always assume there are enough objects to meet the goal conditions. We present Algorithm 1 as an approach to solving this problem by encoding assumptions derived from the goal condition. The algorithm first incorporates instances, attributes, and relations from the current knowledge base in the initial state S_0 (Line 1-3). Then, for each object referenced by the operator, the required number from the operator and the number of known existence are calculated (Line 6-7). The algorithm adds new instances to S_0 until the number meets the user's requirements (Line 8-18). For each added instance (Line 9), fluents inherited from type (Line 11), goal conditions (Line 13) and assumptions (Line 15) are also added into S_0. This algorithm is similar to [1], but in [1]. The difference is that we only add instances to the current state when there are insufficient instances. This makes better use of existing knowledge and can reduce the number of re-planning. Suppose such a scenario: there are three tables in the living room, the robot knows that there is a coke on each table, and the user requests a coke from the living room in the next room. Our algorithm adds three instances to the initial states without adding hypothetical instances. Reference [1] adds a hypothetical instance to the initial states. Whether the planning result is optimal depends on which table the hypothetical instance is added. It is not easy to estimate the cost of acquiring a coke without planning because what actions need to be performed is uncertain without planning. We believe that planning is a better way to determine which object to acquire.

After the goal and initial state are generated, the answer set solver is called to generate answer sets using the union of domain representation, goal conditions and initial fluent set. In the returned answer set, a sequence of actions and fluents that denote expected states before and after execution of actions are obtained.

Algorithm 1 Initial State Construction with Assumptions

Require: Current knowledge K, objects O from command, goal conditions G, assumptions A
Ensure: Initial state S_0
$S_0 = (I, P)$
$I = \{i : i \in K; i \text{ is an instance}\}$
$P = \{p : p \in K; p \text{ is an attribute OR relation}\}$
for each object name $o \in O$ **do**
  t_o = type of o
  n = number condition of o in G
  m = instances' number of o_t in I
  while $n > m$ **do**
    i_o = instance with type t_o
    add i_o to I
    P_t = relations and attributes of t_o in K
    add fluents of i to P by replacing references to t_o in P_t
    P_o = conditions of o in G
    add conditions of i to P by replacing references to o in P_o
    A_o = assumptions of o in A
    add assumptions of i to P by replacing references to o in A_o
    $n = n - 1$
  end while
end for
return S_0

5.2. Execution and Monitor

The components described in the previous sections are employed by the Plan Execution and Monitor component, which is the central coordination component for the command execution. The plan generation and execution follow the traditional "planning-execution-monitor-replan" loop. A simplified control flow for the execution of a planning task is shown in Figure 3.

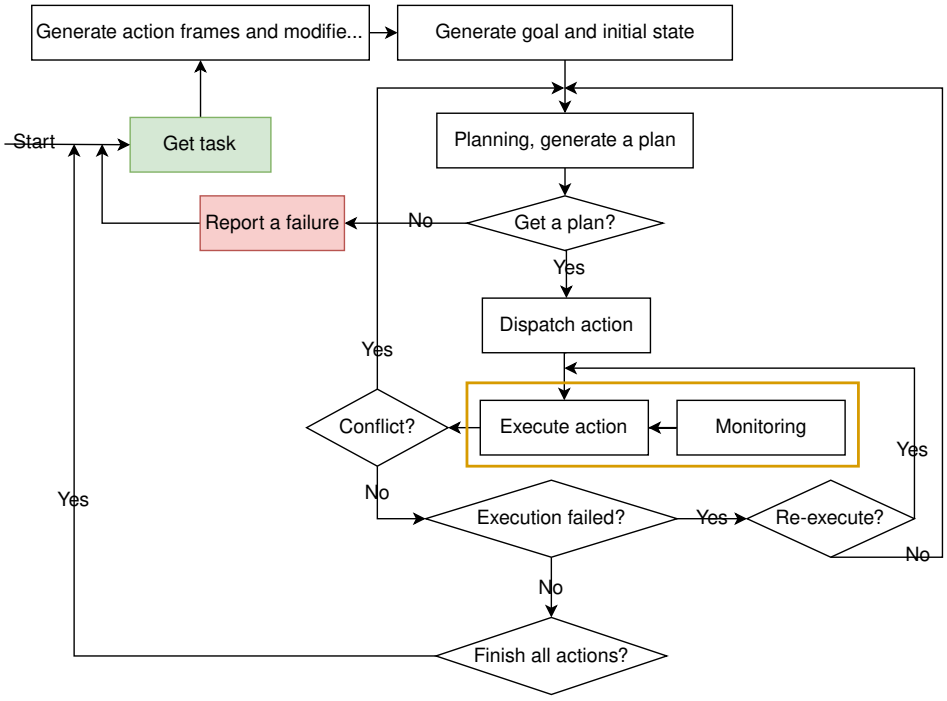

Figure 3. Plan generation, execution, and monitor.

When a command is sent from a person, the NLP module will generate the corresponding *action frames* and *modifiers*, and then construct the goal conditions. Before planning, parameters in action frames will be confirmed. Combined with the current world state, if the specific entity or location information cannot be confirmed, the HRI module will generate a query command to confirm this information and rewrite the goal and state based on the confirmed information. Then, domain knowledge, states, and goals are transmitted to the planner, then the ASP solver generates a plan. If the plan cannot be found, the planner generates feedback indicating the failure. A successful plan includes a series of actions bound variables and the preconditions and effects of these operations, which will be executed one by one. We use CLINGO [42] as our ASP solver.

Each operation corresponding to the symbol planning operator is connected with a state machine that controls the robot's execution. These operations combine elementary or primitive skills into complex skills. The primitive skills are based on services provided by the robot such as inverse kinematics, motion planning, object recognition and location, face recognition, and so forth. For example, motion grasping uses a visual servo to perform object localization accurately.

Action execution may fail due to the uncertainty of perception or the change of environment. To illustrate these changes, the preconditions and effects are verified by the *monitor*. The monitoring program detects changes in the environment and the results of action execution and decides how to make decisions. There are two kinds of environment changes: one is useful information for the current task, and the other is ineffective. Both will be imported into the knowledge database. We divide the useful information into two kinds: the disappearance of preconditions of the actions in the plan; the other is the emergence of new objects or states that meet the task's requirements. In our system, both of them are all seen as conflicts. The former represents that the current plan will not complete the task, while the latter indicates that there may be a better solution to complete the task. They will make the robot re-plan and find a new solution. If new information shows there are no required objects in the knowledge base, the robot will make the new assumptions(assume some required object is somewhere), the same location is only allowed one assumption in a task. In previous work [11], any environment change will trigger a re-plan. We adjust to re-plan when the above two changes occur. The algorithm in [1] re-plans when the unexpected result is detected after the action but ignores subsequent actions (some changes may satisfy current action, but make the conditions of some subsequent actions unsatisfied). Our algorithm's advantage is that it detects whether the environment changes conflict with the current action and detects whether there is a conflict with subsequent actions. As shown in Algorithm 2, according to the action parameters (a) and the action model (D), it instantiates the preconditions (P) of the action, then detects whether these preconditions conflict with environmental changes (line 5-7). In our system, a conflict means that any two states cannot exist simultaneously. These are pre-defined, for example, an object cannot exist in two places simultaneously, and the hand cannot be opened and closed.

If the action is successful, the next action will be performed. Action execution may fail for some reason, so it is necessary to decide whether to re-plan (the maximum number of attempts exceeded or the confirmation operation failed) or try again. When the plan cannot be successfully generated, a failure report is generated to the user. When the task is completed, the robot enters the idle state and waits for a new task.

6. Simulations and Results

To evaluate the system, we have developed a simulation environment with GAZEBO http://gazebosim.org (accessed on 12 February 2021). As shown in Figure 4, the environment contains a domestic house and the model of Kejia. As shown in Figure 5, Kejia is equipped with a wheeled mobile base, a single 5-Degree-of-Freedom arm, a 2D laser range finder, an elevator, a pan-tilt, and an RGBD camera. Our system is implemented as nodes in ROS (Robot Operating System). It subscribes all necessary information (arm joint angles, robot pose, recognized objects, etc.) and publishes control messages (navigate to

Algorithm 2 Conflict Detection

Require: Plan Π, action models D, information changes C
Ensure: Conflict (c, a, p) or not

```
for each a ∈ current action to last action of Π do
    P = instantiate the preconditions of the action a through D
    for each p ∈ P do
        for each c ∈ c do
            if conflict(c, p) then
                return (c, a, p)
            end if
        end for
    end for
end for
return ∅
```

some location, turn on object recognition function to search some object, etc.), which affect the behavior of the robot. The domestic house contains several rooms (bedroom, living room, kitchen, etc.), furniture (bed, table, desk, cupboard, bookshelf, etc.) and some objects (cup, bottle, beer, coke, etc.). For all objects, there are two categories: specific and similar. The specific object has a unique label in the perception system, and this object is bound to this label. Similar objects (apples, same bowl, etc.) share a vision system label. For similar objects, their names in the knowledge base are different. The mapping between the vision system and knowledge base depends on location, size, and other properties.

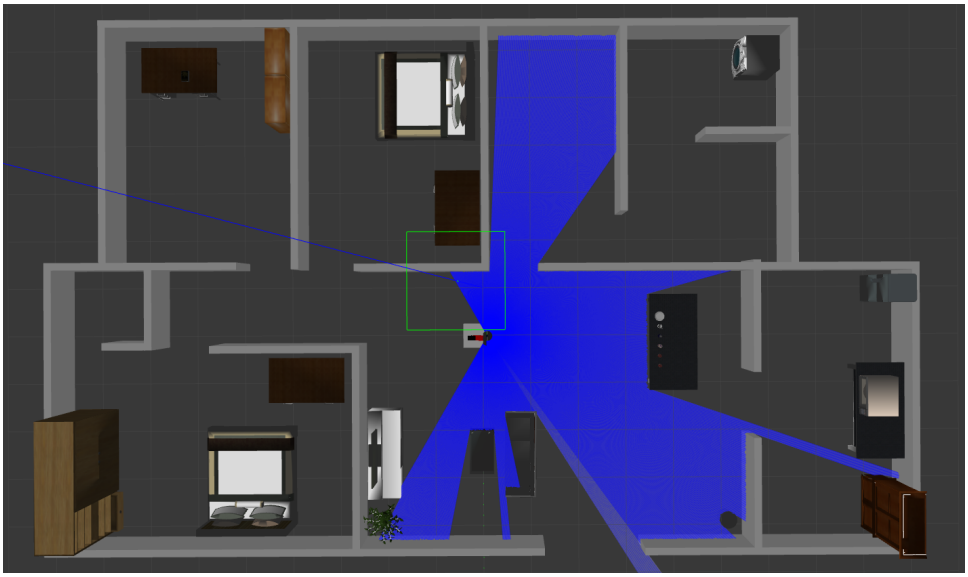

Figure 4. The domestic environment, which includes two bedrooms, one kitchen, one living room, and one study.

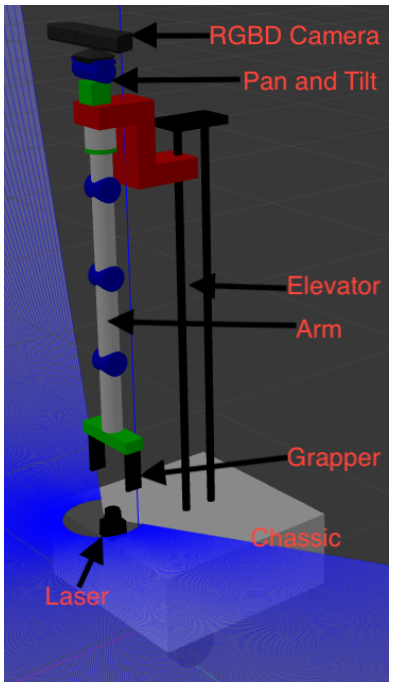

Figure 5. The model of Kejia.

In the experiment, we mainly consider three kinds of uncertainties: HRI uncertainties (vague or erroneous information from users), changing environment (object removed from a known position or object appears in an unexpected place), and execution errors (failed grab or navigation). The required objects and their positions may be unknown to the robot, so the robot needs to assume that there are objects needed in the place given by the user. Here are five scenarios to show how robots respond to these challenges. (A demo video is available at https://youtu.be/y4hJ1OODIG0 (accessed on 12 February 2021).)

Scenario 1 (vague information): Jamie requested a coke from the kitchen. Initially, Jamie and the robot were in the study, the doors from the study to the kitchen are unlocked, the robot knew nothing about coke. The robot got a command "*bring a coke from the kitchen for me*", then denpendencies *root(ROOT-0, bring-1)*, *dobj(bring-1, coke-3)*, *nmod:from(bring-1, kitchen-6)* and *nmod:for(kitchen-6, me-8)* were generated, they assigened the *Action*, *Object*, *Source* and *Goal* of the action frame *action(robot, bring, coke, kitchen, jamie)*, where *"me"* was replaced by the user'name. The robot got vague information: in the kitchen. The robot *first assumed* there was coke in the kitchen table, then started searching in the kitchen by visiting the kitchen table first. There was one cup, one bottle, and one bowl on the kitchen table, but no cokes. The robot made the *second assumption*: the coke was in the cupboard. It visited the cupboard and found two cokes and one beer, then bring one coke to Jamie.

Scenario 2 (error information): Jamie requested a cup from the living table. The robot got a command "*get a cup from the living table for me*", then denpendencies *root(ROOT-0, get-1)*, *dobj(get-1, cup-3)*, *nmod:from(get-1, table-7)* and *nmod:for(table-7, me-9)* were generated, they assigened the *Action*, *Object*, *Source* and *Goal* of the action frame *action(robot, get, cup, living_table, jamie)*, where *"me"* was replaced by the user'name and *living_table* generated based on a modifier *amod(table-7, living-6)*. Jamie and the robot were in the study, and Jamie asked the robot to go to the living table and brought a cup. Though the robot knew a cup on the kitchen table, it first tried to get a cup from the living table. The robot *first assumed* there was a cup in the living table, then started searching by visiting the living table. There was nothing on the living table. The robot *abandoned the information from Jamie*. It got the information that a cup was on the kitchen table in Scenario 1. It visited the kitchen table and found one cup, one water bottle, and one bowl, then bring the cup to Jamie.

Scenario 3 (disappearing target): Jamie requested a coke. The robot got a command "*get a coke for me*", then denpendencies *root(ROOT-0, get-1)*, *dobj(get-1, coke-3)*, and *nmod:for(coke-*

3, me-5) were generated, they assigened the *Action*, *Object*, and *Goal* of the action frame *action(robot, get, coke, none, jamie)*, where *"me"* was replaced by the user'name. Jamie and the robot were in the study, the robot knew there was one coke in the cupboard, it visited the cupboard, and started to search the coke, but did not found any coke. The robot *removed the item from the knowledge base* and made an textitassumption: there was coke on the kitchen table. However, after it reached the kitchen table, it found no coke on the kitchen table. It made *another assumption*: the coke was on the dining table. The robot navigated to the dining table and found two cokes, finally it taken a coke from the dining table and handed it over to Jamie.

Scenario 4 (unexpected target): Jamie requested a coke and a beer. The robot got a command "*bring a coke for me, then bring a beer for me*", similar to scenario 3, two action frame *action(robot, bring, coke, none, jamie)* and *action(robot, bring, beer, none, jamie)* was generated. Jamie and the robot were in the study, the robot knew there was one coke in the dining table and a beer in the cupboard, it visited the dining table, and started to search the coke, then it found coke and a beer. *An unexpected target (the beer)* was found, the robot *added a new item to the knowledge base* and triggered to *replan*, a better solution was generated. The robot brought the coke from the dining table to Jamie and then navigated to the dining table, take the beer from the dining table to Jamie.

Scenario 5 (failed grab): Jamie requested a bowl from the kitchen table. The robot got a command "*get a bowl from the kitchen table for me*", similar to scenario 1, action frame *action(robot, get, bowl, kitchen_table, jamie)* was generated. Jamie and the robot were in the study. The robot known there was one bowl on the kitchen table. It visited the kitchen table and started to search the bowl, and then it found the bowl. While when the robot picked up the bowl from the kitchen table, it *tried twice*, but *both failed*. So it navigated to Jamie and reported this failure.

The demonstration shows that the system can serve under an uncertain and changing environment by involving assumptions and changing detection. By identifying assumptions or detections inconsistent with the knowledge base or not, the robot performs the original plan or makes new assumptions or replans.

7. Conclusion

This paper presents a planning system for a general-purpose service robot by leveraging HRI, assumptions, and continuous sensing, which is aimed at behaving more intelligently in domains containing incomplete information, under-specified goals, and dynamic changes. By combining assumption and symbolic planning, the robot can serve without knowing the required object's position. Proper use of assumptions combined with continuous sensing can help handle unpredictable domain changes and robust behavior in the open world. Our algorithm can find potential conflicts earlier and better balance assumptions and existing knowledge to reduce the cost of the task. In the future, we will address how to make better assumptions to improve planning efficiency.

Author Contributions: Conceptualization, G.C. and X.C.; methodology, G.C.; software, G.C. and W.S.; validation, G.C; formal analysis, G.C.; investigation, G.C.; resources, G.C. and X.C.; data curation, G.C.; writing—original draft preparation, G.C.; writing—review and editing, X.C. and G.C.; visualization, G.C. and W.S.; supervision, X.C.; project administration, G.C. and X.C.; funding acquisition, X.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Natural Science Foundation of China grant number U1613216,61573333.

Data Availability Statement: Not Applicable, the study does not report any data.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Jiang, Y.; Walker, N.; Hart, J.W.; Stone, P. Open-World Reasoning for Service Robots. In proceedings of the International Conference on Automated Planning and Scheduling, Berkeley, CA, USA, 11–15 July 2019, 725–733.

2. Hanheide, M.; Gbelbecker, M.; Horn, G.S.; Pronobis, A.; Sj, K.; Aydemir, A.; Jensfelt, P.; Gretton, C.; Dearden, R.; Janicek, M.; et al. Robot Task Planning and Explanation in Open and Uncertain Worlds. *Artif. Intell.* **2017**, *247*, 119–150. doi:10.1016/j.artint.2015.08.008.
3. Savage, J.; Rosenblueth, D.A.; Matamoros, M.; Negrete, M.; Contreras, L.; Cruz, J.; Martell, R.; Estrada, H.; Okada, H. Semantic reasoning in service robots using expert systems. *Robot. Auton. Syst.* **2019**, *114*, 77–92. doi:https://doi.org/10.1016/j.robot.2019.01.007.
4. Ghallab, M.; Nau, D.; Traverso, P. *Automated Planning: Theory and Practice*; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2004.
5. Gelfond, M.; Kahl, Y. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach*; Cambridge University Press: Cambridge, UK, 2014.
6. Sridharan, M.; Gelfond, M. Using Knowledge Representation and Reasoning Tools in the Design of Robots. Available online: http://ceur-ws.org/Vol-1648/paper12.pdf (accessed on 17 February 2021).
7. Stenmark, M.; Malec, J. Connecting Natural Language to Task Demonstrations and Low-Level Control of Industrial Robots. Available online: http://ceur-ws.org/Vol-1540/paper_05.pdf (accessed on 17 February 2021).
8. Chen, K.; Lu, D.; Chen, Y.; Tang, K.; Wang, N.; Chen, X. The Intelligent Techniques in Robot KeJia - The Champion of RoboCup@Home 2014. In Proceedings of the RoboCup 2014: Robot World Cup XVIII, Joao Pessoa, Brazil, 19–25 July 2014. doi:10.1007/978-3-319-18615-3_11.
9. Chen, X.; Ji, J.; Jiang, J.; Jin, G.; Wang, F.; Xie, J. Developing high-level cognitive functions for service robots. In Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, 10–14 May 2010; pp. 989–996.
10. Chen, X.; Xie, J.; Ji, J.; Sui, Z. Toward Open Knowledge Enabling for Human-Robot Interaction. *J. Hum.-Robot Interact.* **2013**, *1*, 100–117. doi:10.5898/JHRI.1.2.Chen.
11. Chen, K.; Yang, F.; Chen, X. Planning with Task-Oriented Knowledge Acquisition for a Service Robot. In Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, New York, NY, USA, 9–15 July 2016; pp. 812–818.
12. Abukhalil, T.Y.; Patil, M.; Sobh, T.M. UBSwarm: Design of a Software Environment to Deploy Multiple Decentralized Robots. Available online: https://scholarworks.bridgeport.edu/xmlui/bitstream/handle/123456789/546/Tame_Poster_ASEE_2014.pdf?sequence=3&isAllowed=y (accessed on 17 February 2021).
13. Shank, R.C.; Tesler, L. A Conceptual Dependency Parser for Natural Language. In Proceedings of the Third International Conference on Computational Linguistics, Stockholm, Sweden, 1–4 September 1969. doi:10.3115/990403.990405.
14. Culbert, C.; Riley, G.; Donnell, B. *CLIPS reference manual*; Artificial Intelligence Section, Lyndon B. Johnson Space Center: Houston, TX, USA, 1989.
15. Puigbo, J.; Pumarola, A.; Angulo, C.; Téllez, R.A. Using a cognitive architecture for general purpose service robot control. *Connect. Sci.* **2015**, *27*, 105–117. doi:10.1080/09540091.2014.968093.
16. Laird, J.E. *The Soar Cognitive Architecture*; The MIT Press: Cambridge, MA, USA, 2012.
17. Roveda, L.; Spahiu, B.; Terkaj, W. On the Proposal of a Unified Safety Framework for Industry 4.0 Multi-Robot Scenario. In Proceedings of the 27th Italian Symposium on Advanced Database Systems, Castiglione Della Pescaia (Grosseto), Italy, 16–19 June 2019.
18. Antonelli, D.; Bruno, G. Ontology-Based Framework to Design a Collaborative Human-Robotic Workcell. In *Working Conference on Virtual Enterprises*. Springer: Berlin/Heidelberg, Germany, 2017. doi:10.1007/978-3-319-65151-4_16.
19. Talamadupula, K.; Benton, J.; Schermerhorn, P.W.; Kambhampati, S.; Scheutz, M. Integrating a Closed World Planner with an Open World Robot: A Case Study. In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, Atlanta, GA, USA, 11–15 July 2010.
20. Talamadupula, K.; Benton, J.; Kambhampati, S.; Schermerhorn, P.W.; Scheutz, M. Planning for human-robot teaming in open worlds. *ACM Trans. Intell. Syst. Technol.* **2010**, *1*, 1–24. doi:10.1145/1869397.1869403.
21. Petrick, R.P.A.; Bacchus, F. A Knowledge-Based Approach to Planning with Incomplete Information and Sensing. In Proceedings of the Sixth International Conference on Artificial Intelligence Planning Systems, Toulouse, France, 23–27 April 2002; pp. 212–222.
22. Thomason, J.; Zhang, S.; Mooney, R.J.; Stone, P. Learning to Interpret Natural Language Commands through Human-Robot Dialog. In Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, Buenos Aires, Argentina, 25–31 July 2015.
23. Sanelli, V.; Cashmore, M.; Magazzeni, D.; Iocchi, L. Short-Term Human-Robot Interaction through Conditional Planning and Execution. In Proceedings of the Twenty-Seventh International Conference on Automated Planning and Scheduling, Pittsburgh, PA, USA, 18–23 June 2017. doi:10.1007/s12369-019-00606-y.
24. Brenner, M.; Nebel, B. Continual Planning and Acting in Dynamic Multiagent Environments. In Proceedings of the 2006 International Symposium on Practical Cognitive Agents and Robots, Perth, Australia, 27–28 November 2006 doi:10.1145/1232425.1232431.
25. Brenner, M.; Hawes, N.; Kelleher, J.D.; Wyatt, J.L. Mediating between Qualitative and Quantitative Representations for Task-Orientated Human-Robot Interaction. In Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, 6–12 January 2007; pp. 2072–2077.

26. Brenner, M. Situation-Aware Interpretation, Planning and Execution of User Commands by Autonomous Robots. In Proceedings of the 16th IEEE International Symposium on Robot and Human Interactive Communication, Jeju Island, Korea, 26–29 August 2007; pp. 540–545. doi:10.1109/ROMAN.2007.4415145.
27. Lu, D.; Wu, F.; Chen, X. Understanding User Instructions by Utilizing Open Knowledge for Service Robots. arXiv **2016**, arXiv:1606.02877. Available online: https://arxiv.org/pdf/1606.02877.pdf (accessed on 17 February 2021).
28. Hoffmann, J.; Brafman, R.I. Contingent Planning via Heuristic Forward Search with Implicit Belief States. In Proceedings of the Fifteenth International Conference on International Conference on Automated Planning and Scheduling, Monterey, CA, USA, 5–10 June 2005.
29. Son, T.C.; Tu, P.H.; Baral, C. Planning with Sensing Actions and Incomplete Information Using Logic Programming. In *International Conference on Logic Programming and Nonmonotonic Reasoning*. Springer: Berlin/Heidelberg, Germany, 2004; pp. 261–274. doi:10.1007/978-3-540-24609-1_23.
30. Mininger, A.; Laird, J.E. Interactively Learning Strategies for Handling References to Unseen or Unknown Objects. *Adv. Cogn. Syst.* **2016**, *5*, 1–16.
31. Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtós, N.; Carreras, M. ROSPlan: Planning in the Robot Operating System. In Proceedings of the Twenty-Fifth International Conference on Automated Planning and Scheduling, Jerusalem, Israel, 7–11 June 2015; pp. 333–341.
32. Fox, M.; Long, D. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *J. Artif. Intell. Res.* **2003**, *20*, 61–124. doi:10.1613/jair.1129.
33. Petrick, R.P.A.; Gaschler, A. Extending Knowledge-Level Contingent Planning for Robot Task Planning. Available online: https://mediatum.ub.tum.de/doc/1281539/file.pdf (accessed on 17 February 2021).
34. Lim, G.H.; Suh, I.H.; Suh, H. Ontology-Based Unified Robot Knowledge for Service Robots in Indoor Environments. *IEEE Trans. Syst. Man Cybern. Syst. Hum.* **2011**, *41*, 492–509. doi:10.1109/TSMCA.2010.2076404.
35. Aydemir, A.; Pronobis, A.; Göbelbecker, M.; Jensfelt, P. Active Visual Object Search in Unknown Environments Using Uncertain Semantics. *IEEE Trans. Robot.* **2013**, *29*, 986–1002. doi:10.1109/TRO.2013.2256686.
36. Zhang, S.; Khandelwal, P.; Stone, P. Dynamically Constructed (PO)MDPs for Adaptive Robot Planning. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; pp. 3855–3863.
37. Zhang, S.; Sridharan, M.; Gelfond, M.; Wyatt, J.L. Towards an Architecture for Knowledge Representation and Reasoning in Robotics. In Proceedings of the Social Robotics - 6th International Conference, ICSR 2014, Sydney, Australia, 27–29 October 2014. doi:10.1007/978-3-319-11973-1_41.
38. Blount, J.; Gelfond, M.; Balduccini, M. A Theory of Intentions for Intelligent Agents. In Proceedings of the International Conference on Logic Programming and Nonmonotonic Reasoning, LPNMR 2015, Lexington, KY, USA, 27–30 September 2015; pp. 134–142. doi:10.1007/978-3-319-23264-5_12.
39. Srivastava, S.; Zilberstein, S.; Gupta, A.; Abbeel, P.; Russell, S.J. Tractability of Planning with Loops. In Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, Austin, Texas, USA, 25–30 January 2015; pp. 3393–3401.
40. Al-Moadhen, A. Semantic Based Task Planning for Domestic Service Robots. Ph.D. thesis, Cardiff University: Cardiff, UK, 2015.
41. Gelfond, M.; Lifschitz, V. The Stable Model Semantics for Logic Programming. In Proceedings of the Fifth International Conference and Symposium, Seattle, Washington, USA, 15–19 August 1988; pp. 1070–1080.
42. Gebser, M.; Kaminski, R.; Kaufmann, B.; Schaub, T. Clingo = ASP + Control: Preliminary Report. arXiv **2014**, arXiv:1405.3694. Available online: https://arxiv.org/pdf/1405.3694 (accessed on 17 February 2021).
43. Klein, D.; Manning, C.D. Fast Exact Inference with a Factored Model for Natural Language Parsing. In Proceedings of the Advances in Neural Information Processing Systems 15, Vancouver, BC, Canada, 9–14 December 2002; pp. 3–10.
44. Universal Dependencies. Available online: http://universaldependencies.org/docsv1/u/dep/index.html (accessed on 17 February 2021).