The simulation of a service robot for task planning

Guowei Cui, Wei Shuai and Xiaoping Chen

School of Computer Science and Technology, University of Science and Technology of China, 230027 Hefei Anhui, China

Corresponding author's e-mail: xpchen@ustc.edu.cn

Abstract. This paper introduces a simulation for service robots based on Robot Operating System (ROS) and Gazebo. It aims to test and evaluate task planning algorithms. After building the environment and the service robot model, a 2D map is built. To make use of semantic navigation, we construct a topological map from the 2D map. Several functions are written to realize the robot's necessary actions, including navigation, grasping, placing, handing over, and searching person/object. Answer Set Programming (ASP) is used for planning; due to its complicated coding and poor legibility, we define an easy to understand format and translate it into ASP. Finally, we test our task planning system within the uncertain Gazebo environment. The experimental results show the usability of the simulation for task planning tests.

1. Introduction

Task Planning integrates all robot's abilities to make the robot a running system, adapt to dynamic changes, and behave intelligently. In recent years, it has received more and more attention from researchers. Savage et al. [1] use a conceptual-dependency interpreter extracts semantic role structures from the input sentence and planning with the open-source expert system CLIPS [2]. Some work [3] focuses on using open source knowledge to handle incomplete information. Hanheide et al. [4] extend an active visual object search method to explain failures by planning over explicitly modelled additional action effects and assumptive actions. Jiang et al. [5] provide an open world task planning approach for service robots by forming hypotheses implied by commands of operators.

Since testing task planning algorithms on real robots is costly and difficult, an appropriate simulation is necessary. Simulation is a vital part of robot software development. Within the simulation, we can test and evaluate the algorithms without damaging the robot, and we can do some research when we cannot access the robot. This paper builds a simulation for the service robot Kejia to test its task planning algorithm. In fact, there are many simulators [6-8], either commercially available or open-source. We use Gazebo [9] as our simulator. Gazebo is a 3D dynamic simulator, which can accurately and effectively simulate a swarm of robots in complex indoor and outdoor environments. Similar to the game engine which provides high fidelity visual simulation, gazebo provides high fidelity physical simulation, which provides a complete set of sensor models and a very user-friendly interaction mode. Gazebo can simulate complex systems and a variety of sensor components. It is used primarily in developing robots used in interaction, to lift or grab objects, push, or any other activity requiring recognition and localization in space.

Gazebo is also compatible with Robot Operating System (ROS) [10], and anyone can develop a plugin with model components. The main goal of ROS is to provide code reuse support for robot

research and development. ROS is a distributed process (i.e. “node”) framework. These processes are packaged in packages and function packages that are easy to share and publish. ROS supports a joint system similar to code repository, which can realize project collaboration and release. This design enables the development and implementation of a project to be completely independent of the decision-making from the file system to the user interface.

Our simulation contains a domestic environment (including rooms, furniture, appliances, and manipulable objects) and a service robot. The robot model refers to Kejia [11], which has relatively complete atomic ability: navigation, perception, object manipulation, human-robot interaction, task understanding and planning, etc. Kejia represents domain knowledge learned through natural language processing and leverages a symbolic planner for problem-solving and planning to provide high-level functions [12]. The system has been extended to acquire task-oriented knowledge by interacting with the user and sensing the environment [13]. We have enhanced its ability to cope with environmental change and uncertainty in [14], the task planning algorithm used in this work. 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. Answer Set Programming (ASP) [15] is used for task planning by many systems. The contributions of this paper: 1) We build a domestic environment and a service robot model, realize the robot’s necessary actions that can be triggered structured command; 2) We provide a tool to edit the map to generate a semantic map, and the robot uses this semantic map for semantic navigation. We put all these codes at [16].

The remainder of this paper is organized as follows. Section 2 introduces models (robot model and various environmental objects) used in the simulation. Section 3 presents the actions for robot task planning. In section 4, the task planning algorithm is described, and a series of cases are used to validate the simulation system. Section 5 concludes the paper and summarizes the future work.

2. Simulation world and robot modelling

In this section, we introduce the simulation world built and the models in the simulation world. These models include objects, buildings, furniture, and a robot.

2.1. The simulation world

In this simulation, there are various environmental objects. We divided environmental objects into two categories: location and manipulable object. Location is a beacon for navigation; it can’t be moved (as a bathtub) or hardly moved (a bed, for example). In this system, these locations include bed, table, cabinet, wardrobe, refrigerator, oven, etc. The objects often to be moved and manipulated are called manipulable objects. We have four categories of items: food, drink, fruit, and container. The world file specifies the simulation background, lighting, camera pose, physics engines, etc. We create seven rooms, and each room contains some locations and some manipulable objects placed on the locations. The layout of them is shown in figure 1.

2.2. The robot modelling

As figure 2 shows, the robot model is referred to Kejia robot. The robot model comprises a chassis, four wheels (two driving wheels and two driven wheels), an elevator, an arm and a support. All these parts of the model are connected by joints. The robot model is equipped with two driving wheels, which are controlled by a gazebo plugin. Each wheel’s radius is 9cm, and the distance from the central axis of the two wheels is 20cm. A 2d laser sensor is mounted on the chassis. The laser sensor, a gazebo ray sensor with a wide scanning angle of 240 degrees and an update rate of 30 Hz, is used for mapping, self-localization and navigation. The support contains a support base, a neck (pan and tilt) and an RGBD sensor. The pan/tilt and the arm are mounted on the support base. The robot is about 1.6 meters in height; for supporting the real-time environmental perception, the RGBD sensor, a gazebo depth sensor with a resolution of 640x480 and a frame rate of 20 Hz, is mounted on the pan and tilt.

The elevator and support form a lift to adjust the height of the depth sensor and the arm. The five-degree-of-freedom (DOF) arm enables our robot to complete manipulation tasks in the indoor environment flexibly. It has a reach of over 87 centimetres from the centre of the chassis while fully extended. The model has a two-finger gripper, the length of each finger is 10cm. When the gripper is fully opened, the distance between them can reach 16cm. It should be noted that the simulation robot does not rely on physical properties to grasp objects but uses a gazebo plugin (gripper). Because this work is not aimed at grasping research, so this process is simplified.

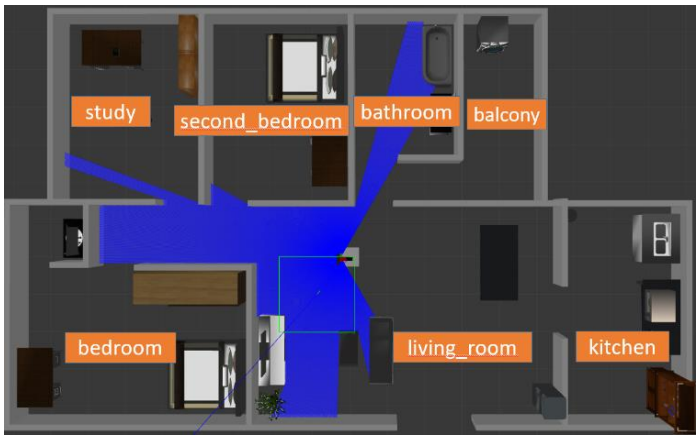

Figure 1. The simulation world.

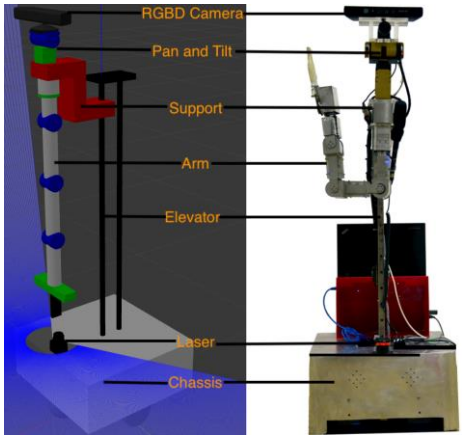

Figure 2. The robot model.

3. Action realization

To realize the actions for task planning, a set of basic motions need to be implemented. We realize these basic motions by *ros_control* [17] package. The *ros_control* software package takes the joint state data as input and set points from the robot actuator encoder. The controller manager sends control command via Joint Command Interfaces to gazebo plugins and gets joint state via Joint State Interfaces from gazebo plugins.

3.1. Basic motions

The basic motions include pan/tilt control, robot rotation & moving, elevator traveling up/down, arm moving, and gripper opening and closing.

3.1.1. Navigation. We use package *move_base* as the implementation of navigation. The *move_base* package provides an implementation of an action that, given a goal in the world, will attempt to reach it with a mobile base. The *move_base* node links together a global and local planner to accomplish its global navigation task. In this work, we use Dijkstra's algorithm as the global planner and Dynamic Window Approach as the local planner. The target pose is sent to the topic */move_base/goal* to trigger the *navigation*.

3.1.2. Ev moving & pan/tilt rotation. The elevator and pan/tilt are controlled by the position controller, an available controller plugin in *ros_control*. To use *ros_control* to control the Kejia model in Gazebo, a transmission element, which is used to bind the actuator to the joint, should be added in the model's URDF (Unified Robot Description Format) file. URDF is an XML specification to describe a robot. The specification of the robot model covers kinematic and dynamic description, visual representation, collision model.

3.1.3. Arm moving & gripper opening/closing. Based on *ros_control*, *open_hand* and *close_hand*, which are used to grasp or release an object, are implemented to open/close the gripper. We use a gripper plugin to simplify the simulation. This plugin fixes an object that is grasped to the robot hand to avoid problems with physics engines and helps the object staying in the robot hand without slipping

out. The action *arm_reach* is defined as accepting a spatial position and moving the arm so that the gripper's centre reaches that position. For a better explanation, according to whether B is higher than A and whether C is higher than A or B, six cases are shown in figure 3. AB, BC and CD are three parts of the arm; A, B and C are the joints. The point D is the target position to reach at the end of the arm. Limited by the robot's arm structure, after moving the arm to the target (x, y, z), these four points are always on a same plane. The last part of the arm (CD in figure 3) always needs to be parallel to the ground. Use α, β, and γ to represent the angles of the three joints (A, B, C) need to be rotated. Some shorthand symbols are introduced by equation (1), where l_1, l_2, l_3 and (x, y, z) are known.

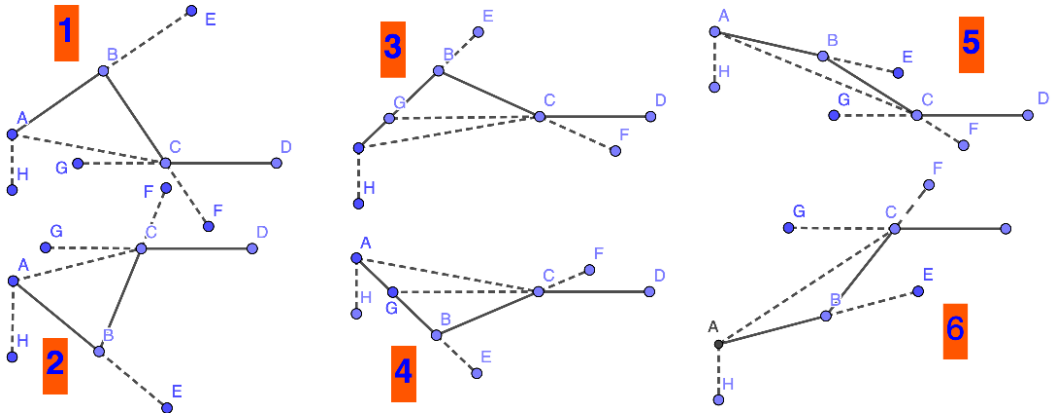

Figure 3. Arm fetching cases.

$$d_{xy} = \sqrt{x^2 + y^2},\ d = \sqrt{x^2 + y^2 + z^2},\ l_1 = AB,\ l_2 = BC,\ l_3 = CD,\ l_4 = AC,$$
$$\delta = \angle CAB,\ \zeta = \angle ABC,\ \eta = \angle BCA. \tag{1}$$

According to whether z is bigger than 0 (D above A or not), we get equation (2):

$$\theta = \arctan(z / l_4) = \begin{cases} \angle GAC & z \geq 0 \\ -\angle GAC & z < 0 \end{cases} \tag{2}$$

Taking case 1 as an example, we can obtain equation (3) according to triangles' properties.

$$l_4 = \sqrt{(d - l_3)^2 + z^2},\ \eta = \pi - \zeta - \delta,$$
$$\zeta = \arccos((l_1^2 + l_2^2 - l_4^2) / (2 * l_1 * l_2)),\ \delta = \arccos((l_1^2 + l_4^2 - l_2^2) / (2 * l_1 * l_4)). \tag{3}$$

In the project, the angle obtained by rotating up the joint is positive; otherwise, it is negative. So we get equation (4).

$$\alpha = \angle HAB = \pi / 2 + \delta + \theta,\ \beta = -\angle CBE = \zeta - \pi,\ \gamma = \angle FCD = \eta - \theta. \tag{4}$$

Similarly, cases 3 and 5 get the same equation representation as 1. Nevertheless, cases 2, 4 and 6 get another representation of the equation, as shown in equation (5). The final result selects one of the two results according to the constraint range of the angles.

$$\alpha = \angle HAB = \pi / 2 - \delta + \theta,\ \beta = \angle CBE = \pi - \zeta,\ \gamma = \angle FCD = -\eta - \theta. \tag{5}$$

3.2. Actions

This section describes the actions used for task planning as following.

- *pickup(O,L,G)*. The robot grabs the object (O) from the designated place (L) to the hand (G). This process is divided into four steps: a) searching for the object and confirming its position;

b) adjusting the pose of the robot to make it convenient for further observation and grasping; c) observing the pose of the object and calculating the translation, elevator height and all joints angles of the arm required for grasping; d) moving the robot and the arm to complete the grasping.

- *putdown(O,L,G)*. The robot places the object (*O*) from the hand (*G*) to some location (*L*). This action consists of three steps: observe the location and confirm the position to be placed; calculate the translation amount and arm joint angle required for placement; move the robot and arm and release the object.
- *handover(O,H,G)*. The robot hands over the object (*O*) to the person (*H*) from its gripper (*G*). The robot reminds people to catch the object after a fixed time to release the object, and leave the rest of the work to the person.
- *findObj(O,L)*. The robot observes whether an object (*O*) is found at the specified location (*L*).
- *findPerson(H,L)*. The robot observes whether the person (*H*) can be found at the location (*L*).
- *moveIn(R_2,R_1,D)*. The robot passes through the designated door (*D*) and moves from one room (R_1) to another (R_2). This action is based on a semantic map, generated from a normal 2D map by a map editor tool. As shown in figure 4, rooms are connected by doors. As shown in figure 5, the map editor is used to edit the 2D map to generate topological information. The map editor has four types of entities: room, furniture, way, and door. Doors are open by default. The action is divided into two parts: moves to a point in front of the centre of the door and then goes through the door. So the robot can get through the door better, especially when the door is not very wide.
- *moveTo(L,R)*. The robot moves to the special location (*L*). This location and the robot are both in the same room (*R*) in the process.

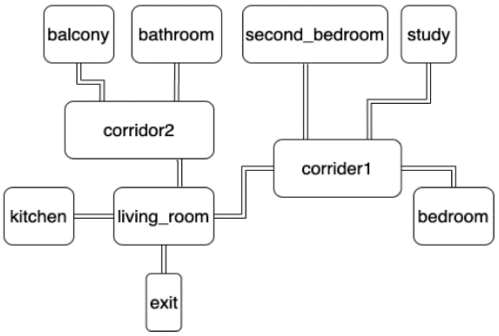

Figure 4. The topological graph.

Figure 5. The map editor.

Table 1. Preconditions and effects.

Preconditions	**Effects**
empty(G), graspable(O), isNear(robot,L), isPlaced(O,L)	*isHeld(O,robot), inHand(O,G), ¬empty(G), ¬isPlaced(O,L), ¬inRoom(O,R) if inRoom(robot,R)*
inHand(O,G), isNear(robot,L), isPlacement(L,true)	*isPlaced(O,L), empty(G), ¬inHand(O,G), ¬isHeld(O,robot), inRoom(O,R) if inRoom(robot,R)*
inHand(O,G), isNear(robot,H), human(H)	*isHeld(O,H), empty(G), ¬inHand(O,G), ¬isHeld(O,robot)*
assume(isPlaced(O,L)), object(O), isNear(robot,L)	*isPlaced(O,L), ¬assume(isPlaced(O,L)), inRoom(O,R) if inRoom(robot,R)*
assume(isNear(H,L)), human(H), isNear(robot,L)	*isNear(H,L), ¬assume(isNear(H,L)), inRoom(H,R) if inRoom(robot,R)*
door(D,R_1,R_2), open(D), inRoom(robot,R_1)	*inRoom(robot,R_2), ¬inRoom(robot,R_1), ¬isNear(robot,L) if isNear(robot,L)*
inRoom(robot,R), inRoom(L,R), not isNear(robot,L)	*isNear(robot,L), ¬isNear(robot,L_1) if isNear(robot,L_1)*

Their preconditions and effects in task planning are listed in table 1 (actions correspond to rows 2 through 8 in order). These actions require more basic operations provided by the basic operations described in section 3.1.

4. Task planning testing

In this section, the task planning algorithm is described, and a series of cases are used to validate the simulation system.

4.1. Task planning

We introduced ASP [15] as the knowledge representation and reasoning tool for Kejia, and CLINGO [18] as our ASP solver. ASP is a logic language with Prolog-like syntax and the stable model semantics, and a non-monotonic reasoning mechanism. To run a planner, the system needs initial states, an action domain, and target states (goals). As described in table 1, the action domain is designed according to each action's preconditions and effects. The initial states are generated from the knowledge base, which stores the world's current state and the robot. The robot obtains this information through perception and interaction. The goals are generated from action frames, which are generated from the speech by NLP (Natural Language Processing). An action frame is composed of five parts: Actor, Action, Object, Source, Goal. Actor represents the executor of the action, usually refers to the robot, Action refers to the type of operation, Object is the object to be operated, Source refers to the starting position of Object, and Goal refers to the last position of Object. For each input sentence, the NLP module works in three steps: 1) Parsing, in which Stanford Parser 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 to generate plan. The system performs plan using the classical "plan-execute-monitor-replan" loop. It checks if the change conflicts with actions in the plan, not only the action being performed, but also the subsequent actions, so the conflict can be detected and handled as early as possible. The 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; 3) continuous sensing and conflict detection mechanism that capture dynamical changes in the environments and trigger special processing. A video demo, https://youtu.be/TelJpWJe8q8, shows how the robot deal with the changes in the open world. For more details about the planning algorithm, please refer to [14].

4.2. Serving drinking task

This section details the behaviour of the simulator as it executes a serving drink task. Figure 6 (video demo: https://youtu.be/dxRshRKXQOE) shows robot actions, important knowledge updates, and plan changes along with frames.

The task was initiated by asking the robot to "Bring a beer from the kitchen for me." The NLP module parsed the command and then translated the parsed command into the action frame action(robot, get, beer, kitchen, jamie) and a modifier number(1, beer), where Jamie was the user. Initially, Jamie and the robot were in the study; the doors from the study to the kitchen were unlocked. The robot got vague information: in the kitchen. The module Goal Generation queried the knowledge base's status to find if there were enough beers in the kitchen. At first, in the database, there was no beer in the kitchen. Then an assumption was introduced: assume there was a beer on the kitchen table. The goal and assumption were generated in ASP format, as shown in table 2.

Table 2. Goal and assumption.

ASP	Description
1 { goal(isHeld(A1,jamie)) : requied(A1,a3), not init(isHeld(A3,jamie))} 1	Goal
init(assume(isPlaced(o_a3_1,kitchen_table))), is(o_a3_1,beer), required(o_a3_1,a3)	Hypothesis

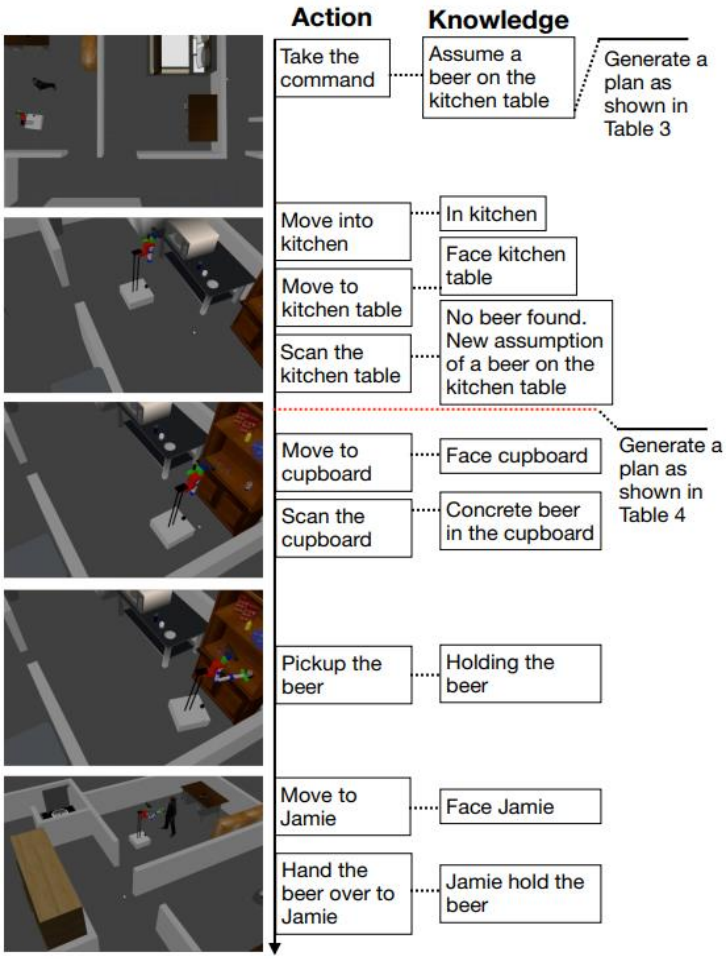

Figure 6. The timeline of the execution for command: "Bring a beer from the kitchen for me".

Table 3. Actions in the plan.

Action	Description
moveIn(corridor1,study,study_corridor1)	Move to corridor1 from study
moveIn(living_room,corridor1,living_room_corridor1)	Move to living_room
moveIn(kitchen,living_room,living_room_kitchen)	Move to kitchen
moveTo(kitchen_table,kitchen)	Move to the kitchen_table
findObj(o_a3_1,kitchen_table)	Search beer from kitchen_table
pickup(o_a3_1,kitchen_table,gripper)	Pickup beer from kitchen_table
moveIn(living_room,kitchen,living_room_kitchen)	Move to living_room
moveIn(corridor1,living_room,living_room_corridor1)	Move to corridor1
moveIn(study,corridor1,study_corridor1)	Move to study from corridor1
moveTo(jamie,study)	Move to Jamie
give(o_a3_1,jamie,gripper)	Hand the beer over to Jamie

The Planner generated a plan to achieve the goal from the initial state. This plan consisted of 11 actions, as shown in table 3. The robot moved to the kitchen table and started searching in the kitchen by visiting the kitchen table. There was one cup, one bottle, and one bowl on the kitchen table, but no beers. The robot detected this conflict and then made a new assumption: "the beer is in the cupboard." Based on the new assumption, a new plan was generated, as shown in table 4. It visited the cupboard,

found two cokes and one beer in the cupboard, grabbed the beer, moved to Jamie, and handed it over to Jamie. By confirming whether the environment is consistent with the knowledge base or assumptions, the robot performs the original plan or makes new assumptions to trigger a new plan.

Table 4. Actions in the new plan.

Action	Description
moveTo(cupboard,kitchen)	Move to the cupboard
findObj(o_a3_1,cupboard)	Search the beer from the cupboard
pickup(o_a3_1,cupboard,gripper)	Pickup the beer from the cupboard
moveIn(living_room,kitchen,living_room_kitchen)	Move to living_room from kitchen
moveIn(corridor1,living_room,living_room_corridor1)	Move to corridor1 from living_room
moveIn(study,corridor1,study_corridor1)	Move to study from corridor1
moveTo(jamie,study)	Move to Jamie
give(o_a3_1,jamie,gripper)	Hand the beer over to Jamie

5. Conclusions

We have presented a domestic simulation environment that consists of rooms, furniture, and manipulable objects, built a service robot model with URDF. The simulation aims to test task planning algorithms. Several functions are written to realize the robot's basic actions, including navigation, grasping, placing, handing over, and searching person/object. We put the codes at [16] and hope it can be helpful for related research. Through the simulation experiment, the planning algorithm can be well evaluated in various situations. This simulation system can be easily adjusted to develop other task planning algorithms through the actions we have implemented. Overall, the simulation system can facilitate the research on task planning. Further study will focus on modeling more uncertainty (including actions and environment) and improving the simulation speed.

6. References

[1] Savage J, Rosenblueth D A, Matamoros M, Negrete M, Contreras L, Cruz J, Martell R, Estrada H and Okada H 2019 Semantic reasoning in service robots using expert systems *Robotics Auton. Syst.* **114** 77-92

[2] Giarratano J C 1993 Clips user's guide *NASA Technical Report* (Lyndon B Johnson Center)

[3] Chen X, Xie J, Ji J and Sui Z 2013 Toward open knowledge enabling for human-robot interaction *JHRI* **1** 100-17

[4] Hanheide M et al. 2017 Robot task planning and explanation in open and uncertain worlds *Artif. Intell.* **247** 119-50

[5] Jiang Y, Walker N and Stone P 2019 *Proc. of the 29th Int. Conf. on Automated Planning and Scheduling* (Berkeley) pp 725-33

[6] Rohmer E, Singh S and Freese M 2013 *2013 IEEE/RSJ Int. Conf. on Int. Robots and Systems* (Tokyo) pp 1321-6

[7] Michel O 2004 *Int. J. Adv. Robot. Syst.* **1** 5

[8] Carpin S, Lewis M, Wang J, Balakirsky S and Scrapper C 2007 *2007 IEEE Int. Conf. on Robotics and Automation* (Roma) pp 1400-5

[9] Koenig N and Howard A 2004 *2004 IEEE/RSJ Int. Conf. on Int. Robots and Systems* (Sendai) pp 2149-54

[10] Quigley M, Conley K, Gerkey B, Faust J, Foote T, Leibs J, Wheeler R and Ng A Y 2009 *ICRA workshop on open source software* (Kobe) p 5

[11] Chen K, Lu D, Chen Y, Wang K, Wang N and Chen X 2014 *18th Annual RoboCup Int. Symp.* (João Pessoa) pp 130-41

[12] Chen X, Ji J, Jiang J, Jin G, Wang F and Xie J 2010 *9th Int. Conf. on Autonomous Agents and Multiagent Systems* (Toronto) pp 989-96

[13] Chen K, Yang F, Chen X 2016 *Proc. of the 25th Int. Joint Conf. on Artificial Intelligence* (New York) pp 812-8
[14] Cui G, Shuai S and Chen X 2021 Semantic task planning for service robots in open worlds *Future Internet* 13(2)49
[15] Gelfond M and Lifschitz V 1988 *Logic Programming: Proc. of the 5th Int. Conf. and Symp.* (Seattle) pp 1070-80
[16] https://github.com/cuigw1/simulation
[17] http://wiki.ros.org/ros_control
[18] Gebser M, Kaminski R, Kaufmann B, and Schaub T 2014 Clingo = asp + control: preliminary report *arXiv* abs/1405.3694

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