China Robot Competition-RoboCup China Open
Service Robot Intelligence Challenge
2009 Competition Rules

December 2, 2009

Contents

Chapter 1

Introduction

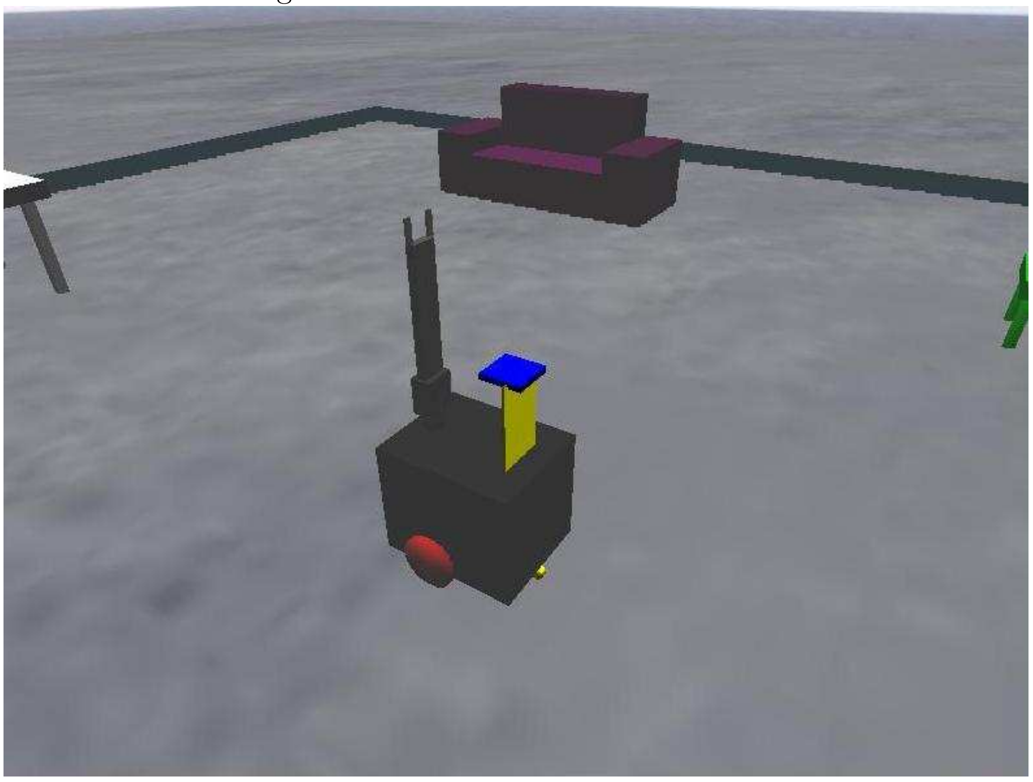
Service robot Intelligence Competition aims at the investigation of high-level functionality of service robots, currently including human-robot communication, automated planning and reasoning. An abstract 3D model simulating the service robot will be provided. A simulated in-door environment is used as the test bed. Human-robot communication is abstracted as text-based (????) natural language description of tasks and command instructions for task execution. The robot sensing of problem domain is stored as files according to certain format (????).

The challenge tasks are designed along the lines of typical indoor applications of an autonomous robot. Each competition task consists of an problem domain description and a task specification. The problem domain description provides initial states of the problem domain, while the task specification describes users' task requirements by specifying goals and constraints along with relevant extra information. A database is used to inform the competing robot the problem domain description. Tasks can be specified in natural language or instruction language. This leads to the two competition programs, the Natural Language Program and the instruction Language Program.

A simulated 3D robot is used in the competition, as shown in Figure 1.1. It can perform a set of atomic actions, fixed to all problem situations. Therefore, the competition can be considered as problem solving based on a fixed set of atomic actions. The robot has two moving wheels; one arm with one hand that can hold only one item at a time; one plate that can hold one object at a time. The robot has basic movement, abilities of picking an item up and putting it down. Based on these basic functionalities, the competition is focusing on human-robot communication, automated planning and reasoning.

The competing program is required to automatically generate a sequence of atomic actions in the specified time frame based on the problem domain

Figure 1.1: 3D Simulated Robot Model



description and task specification, for every challenge task posed by the competition program. The competing robots will be ranked according to the aggregated performance of the generated action sequence for each challenge task.

For testing and evaluating purposes, a simulation problem domain is provided to visualize the 3D problem domain and the execution of the action sequences. The following chapters will provide further details on the rules, ranking criteria, competition program, and interface to the competition platform. The rules are set and can be further explained by the competition committee, currently consisting of Chen Xiaoping, Ji Jianmin (main contact: `jizhen[at]mail.ustc.edu.cn`), Jiang Jiehui, Jin Guoqiang and Xie Jiongkun.

Chapter 2

Basic Rules

2.1 problem domain Description

problem domain description specifies the current situation that the robot is currently facing, including the types of objects appeared in the problem domain, their positions and other attributes. It also provides the current states of the robot.

In this competition, the problem domain description provides:

- Size: the size of the problem domain remains the same throughout the competition (`10m x 10m`).
- Objects: Each object in the problem domain has its own unique ID, type, location and other attributes. Apart from the ID, all other fields are optional. Some objects may not have all properties provided. There is one special object, Human (`sort: human`), representing the user. There is one and only one user in the problem domain. The user's state remains constant during the entire problem solving and task execution process. All other objects are described according to:
  - `sort`: the type of the object. For example, `book, cup, bottle, table, chair` and etc. Please see a list of objects in Appendix A.1.
  - `color`: the color of the object, can be taken from the following set: {`white, red, green, yellow, blue, black`}.
  - `size`: the size of the object, can either by small or big. The size is mainly to differentiate objects that can or cannot be moved by the robot:

    * *Big objects*: objects that cannot be moved by the robot, therefore, its location and states will remain the same throughout the challenge task.
    * *Small objects*: objects that can be moved by the robot, therefore, its location can change. Most likely, they are placed on top of a big object.
  - location: is an integer, a unique ID is specified to each location. It is also assumed that, any two locations are reachable. In other words, the robot can move freely from location X_1 to location X_2. One or more small objects can be placed at one location, but only one big object for one location.
  - robot state: The robot has one hand and one plate. Whether they are holding an object determining the state of the robot. Details include:
    * `location`: the location of the robot, using the same location description ID as described above for the object locations. The object holding by the hand or the plate assume the same location as the robot.
    * `plate`: the state of the robot plate, can take a value of either `empty` or an ID of a small object
    * `hold`: the state of the robot hand, can either be `empty` or an ID of a small object.

The problem domain description is stored as a file, Chapter 3 will provide details on the file format. Problem solving often requires not only the initial states, but also some domain knowledge, for example, one object can only assume one location, and the objects placed in the plate will change location as the robot moves. Such domain knowledge (or background knowledge) needs to be provided by the competition teams.

Here we have only provided a description of the problem domain description file, a formal description can be done using Planning Domain Definition Language (PDDL) [1], please refer to Appendix B.1. During the competition, the problem domain description file will be provided to the competing robot, details please see Competition Platform.

[1]A standard language for planning domain description, often used in International Planning Competition `http://planning.cis.strath.ac.uk/competition`

2.2 Atomic Actions of the Robot

The atomic actions of the robot are an abstraction of the basic functionalities of the robot. Atomic actions are described using pre-conditions and post-conditions. Pre-conditions are a set of constraints in the problem domain, and post-conditions are the changes and the effects in the problem domain. We assume the robot can only affect the problem domain using its atomic actions.

Competing program needs to generate a plan in a specified time frame to accomplish the specified challenge task. The generated plan consists of a sequence of atomic actions. The Competition Platform will evaluate the performance of the plan against the specified task and score the results accordingly. Detailed scoring rules will be explained in the following chapters.

In this competition, there are five atomic actions as explained below, which stays the same throughout the competition.

- *move*(X): The robot moves and arrives at location X. The precondition is that the robot is not at location X [2]. The post condition is that the robot arrives at location X. Other properties that are not affected by this action will remain the same [3].

- *catch*(A): The robot picks object A up. Precondition: object A is in the reachable range (i.e. the robot and object A are at the same location), object A is moveable, and the robot hand is `empty`. Postcondition: the robot hand is holding object A in its hand. All other properties remain unaffected.

- *putdown*(A): The robot puts down object A. Precondition: the robot is holding object A in its hand. Postcondition: the robot hand is `empty`. All other properties remain unaffected [4].

- *toplate*(A): The robot puts object A in its plate. Precondition: the robot is holding object A in its hand, and the plate is `empty`. Postcondition: the robot hand is `empty`, object A is in its plate. All other properties remain unaffected [5].

[2]We assume that all locations on the map are reachable.

[3]The object held in its hand or placed in its plate will move to the same location as the robot moves.

[4]The location of A remains the same as the robot's current location.

[5]The location of A remains the same as the robot's current location, and will move along with the robot.

- *fromplate*(A): The robot picks object A up from its plate. Precondition: object A is in the plate and the robot hand is `empty`. Postcondition: The robot plate is `empty` and the robot hand is holding object A. All other properties remain unaffected.

Again here we only provided the description, for formal PDDL statements please see Appendix B.2.

The competing program should output a sequence of these atomic actions, detailed interface please see Chapter 3.

2.3 Task Description

Human Robot Communication is one of the primary focus of this Service Robot Intelligence Challenge. In the competition this year, the competing program needs to understand user specified task description, calculating the corresponding plan. Human Robot Communicate is carried through different ways of describing the tasks.

Tasks can be specified using three means: *Goal*, *Constraints* and *Extra Information.* The goal specified by the user represents the task that the robot needs to accomplish. For instance, “Give me the green bottle.”, expressing the user’s desire that the “green bottle” should be at the same location as the user `human`. Constraints specify the conditions that must be satisfied during the process of task execution, or the conditions in the problem domain. For example, “Do not catch any bottle on the table.”, specifies that during the task execution, the robot should be not pick the “bottle” on the “table” up. User can also specify some extra or supplementary information to the initial state of the problem domain. For example, “The red bottle is on the table.” Therefore, even if the location of the “red bottle” may not be specified in the initial domain specification, following this information, we can readily deduce the location of the “red bottle” according to the location of the “table”.

During the first level of the competition, domain specification is complete with respect to the specified task. In addition, the task specification will have no constraints and extra information. In the second level, user specified constraints and extra information will be added. In addition, some problem and task specification will be incomplete. Other information will be needed to complete the specified task.

Tasks can be specified in two ways, which corresponding to the two challenge programs, one using natural language with limited vocabulary, the other using pre-defined instructions.

2.3.1 Command-based Interaction

In this program, user specifies the task using instructions in the following set.

First of all, *sort* is defined as one of the types available in the object type list (see Appendix A.1).

The object *obj* in the instruction is defined as:

adj := big | small | white | black | red | green | yellow | blue.
obj := $sort$ | adj $sort$.

The final goal of the task *task* in the instruction is defined as (obj_1 and obj_2 are objects in the instruction):

$task$:= give(human, obj_1) | puton(obj_1, obj_2) | goto(obj_1) |
putdown(obj_1) | catch(obj_1).

The extra information $info$ in the instruction is defined as:

$info$:= on(obj_1, obj_2) | near(obj_1, obj_2) | onplate(obj_1).

The constraints *cons* in the instruction is defined as:

$cons$:= not $task$ | not $info$ | not not $info$.

The instruction *ins* is defined as *ins* := *task* | *info* | *cons*. The instructions used in the competition is a set of *ins*.

The meaning of *task* is further explained below:

- give(human, obj): give the obj to the user, where obj is a small object. The obj should be at the same location as the user, and not in the robot hand nor its plate.
- puton(obj_1, obj_2): put obj_1 onto obj_2, where obj_1 is a small object. obj_1 should be at the same location as obj_2, and not in the robot hand nor its plate.
- goto(obj): the robot moves to where the obj is.
- putdown(obj): the robot drops down obj. obj is a small object,and obj is no longer in the robot hand.
- catch(obj): the robot picks obj up. obj is a small object and obj should be in the robot hand.

info is explained below:

- on(obj_1, obj_2): obj_1 is on obj_2. In other words, obj_1 and obj_2 are at the same location.
- near(obj_1, obj_2): obj_1 is beside obj_2. In other words, obj_1 and obj_2 are at the same location.
- onplate(obj): obj is on the robot plate.

cons is further explained below:

- not *task*: No matter what state the environment and the robot is in, action specified in $task$ is forbidden.
- not *info*: No matter what state the environment and the robot is in, the condition specified in $info$ needs to avoided.
- not not *info*: No matter what state the environment and the robot is in, the condition specified in $info$ needs to be maintained.

2.3.2 Natural Language Based Interaction

In the Natural Language Based Interaction League, tasks are specified using natural language with limited vocabularies. For example, "Give me the red bottle which is near the green bottle.", "The white cup is on the table.","The cup which is white is on the table.", and "There must be a bottle on the desk.".

Table 2.1 has specified a list of allowed vocabulary in the competition [6].

As we can see, the vocabulary listed in Table 2.1 is quite limited. This is to reduce unnecessary complex scenarios in the environment. There is only one pronoun `me`, specifically referring to the only user (in the scenarios, there is one and only one `human`). Not allowing other pronouns helped avoiding the difficult task of pronoun resolution in natural language. In order to allow for adjective clauses, `which` is added to the list of prepositions. The adjectives are used to describe the properties of the objects in the environment. Noun `plate` is used to specifically refer to the plate on the robot. Using the vocabulary, the user can specify tasks "Put the red bottle on the plate.". Constraints are often specified using `must`, and `do not`.

[6] Here we only listed the root format of verbs and nouns, other tense and corresponding plurals are all acceptable.

Table 2.1: Natural Language Limited Vocabulary

Part of Speech	Word
Pronoun	me
Article	a, an, the
Transitive Verb	must
Verb	do, be, give, put, go, catch
Preposition	on, near, next, to, down, which
Adjective	white, black, red, green, yellow, blue, big, small
Adverb	there, not
Noun	plate, all defined object types

So long as the task specification make use of the above vocabulary and follow normal English grammar, there is no extra requirement on the language.

A specified task is normally expressed using imperative sentences. For example, "Go to the table.", "Catch the bottle which is on the chair.". The active verbs are explained in terms of the requirement on object locations and the robot state as shown below, where A and B are referring to objects.

- give: normally used in the form of "give somebody A" or "give A to somebody". As a task, it requires that after the completion of the task, object A is at the same location as "somebody" and no longer in the robot hand nor the plate.

- put: normally used in two forms. One is "put A on/near/next to/down to B". After completing the task, object A should be at the same location as B (B's location remain unchanged), and no longer in the nor the plate. The other is "put A down" or "put down A". After completing the task, object A is no longer in the robot hand.

- go: normally used in the form of "go to A", which requires the robot move to and arrive at where A is at.

- catch: normally used in the form of "catch A", which requires that the robot is holding object A.

Extra information is open specified using narratives. For example, "The book is on the table.". Normally in the form of "A is on/near/next to B" or"There is A on/near/next to B". This can be interpreted as A and B are at the same location, expect that when B is the `plate`. In the case when B is the `plate`, it is interpreted as A is the robot plate.

Constraints are specified using “Do not …”, “There must (not) be …” or“…must (not) be …”. “Do not” is often followed by a task. For example, “Do not go to A”, “Do not put A on B”. “There must (not) be” is often followed by state descriptions. For exaple, “There must (not) be A on B”. “…must (not) be …” often describe the relative relations of two objects. For example, “The red bottle must (not) be on the plate”.

Chapter 3

Scoring Criteria

The service robot challenge requires the competing program to generate a sequence of atomic actions to complete every task specified by the competition platform, following the domain description and task specification. The competition platform will score the competing program according to the aggregated performance of the generated sequence of atomic actions for each task specification.

The performance of the atomic action sequence is judged according to the closeness to the completion of tasks and the number of actions involved.

The closeness to the task completion is determined by the number of tasks it accomplished and the number of constraints it satisfied. A task specification may have multiple task goals or constraints (extra information not considered). An atomic action sequence is considered to have completed a specified task or satisfied a constraint, when

- The final state of the action sequence: Normally, starting from the initial state of the domain, the first atomic action can be performed (i.e. the precondition of this action is satisfied). After successful execution of this action, the system enter into a new state. Then the atomic action that follows is also executable and the same checking goes on until the final sate is reached as we reach the end of the action sequence. If next action can not be executed, then the current state is considered as the final state.
- Completion of a task: The action sequence is considered to have completed a task if the final state meets the task specification.
- Satisfied a constraint: From the initial state to the final state, every step of the sequence execution met the requirement of the constraint.

Scoring criteria is defined as following:

- 10 marks for completing a task.
- 5 marks for maintained one constraint.
- -1 mark for each atomic action.

Therefore, the score for an atomic action sequence for one question is defined as:

$$\text{Question Score} = 10 \times \text{number of completed tasks}$$

$$+5 \times \text{number of maintained constraints} - \text{length of the action sequence}.$$

There are two stages of the competition, each stage will have a group of questions. The competing programs are ranked according to the aggregated total score for all questions.

Chapter 4

The Challenge Platform

The ChallengeServer (Version 1.1) will be used for this competition, source code of the server is available at

`http://www.wrighteagle.org/rco/rco09/`

ChallengeServer is responsible for the management of the question bank (domain description and task specification). During the competition, it will invoke the competing program to solve the specified problem in a specified time frame (in this case, 2 seconds), and evaluate the performance to provide a score of the planning output. At the end of the competition, it will rank all competing programs according to their total score. A C++ interface, inherited from `Plug` class is required from the competing program to implement the corresponding interface methods and generate dynamic link library files (dll files). ChallengeServer uses dynamic link library files to invoke the competing programs.

There are two stages of the competition, each requires the competing program to solve a set of questions based on domain description and task specification. The question bank will not be available before the competition, instead, the questions will be released as a whole at the end of the challenge. The participating teams are strongly suggested to prepare their own question banks according to the format specified in Chapter **??**.

The challenge will be based on Window XP operating system, and the hardware capacities used in this competition are:

- CPU: AMD Athlon(tm) II X4 620

- Memory: 2GB

4.1 Platform Dependency

ChallengeServer runs on Windows XP, and is developed using Visual Studio 2008ś Visual C++. At the moment, it only supports Windows system that has VS2008 installed. In other words, to run this platform, one will have to ensure that both VS2008 and Framework3.5 are installed first.

4.2 Interface

ChallengeServer invokes the competing programs though the dynamic link libary files generated themselves. The competing program should inherit the `Plug` class, and implement the corresponding interface method. You may want to refer to the code under the `sample` folder in the ChallengeServer source code release. Further details are explained below:

1. include header file `planner/plug.h` and create a subclass of the `Plug` class.
2. use the team name as the constructor argument for `Plug`.
3. overload `Plan()` method, to implement the planning algorithm. For each question, the ChallengeServer will call this `Plan()` method and give it a calculation time of 2 seconds.
4. Through `GetTestName(const char* & dname, const char* & tname)` method, you can obtain the file name for the current domain description `dname` and task specification `tname`.
5. Follow a sequential call to the following methods, to output the planning result:
   - action $move(X)$ corresponds to `Move(unsigned int x)`.
   - action $catch(A)$ corresponds to `Catch(unsigned int a)`.
   - action $putdown(A)$ corresponds to `PutDown(unsigned int a)`.
   - action $toplate(A)$ corresponds to `ToPlate(unsigned int a)`.
   - action $fromplate(A)$ corresponds to `FromPlate(unsigned int a)`.
6. `Init()` can be overloaded to provide team specific initialisation if so desired.

7. `Fini()` can be overloaded to delete data if so desired.

8. Using the corresponding class name as the input argument, Macro EXPORT can be used to package the project into Dynamic Link Library, so that it can be invoked by the challenge platform.

4.2.1 Format of the Domain Description

During the competition, the domain description will follow the following format:

First, the objects in the environment will be assigned a unique positive integer, denoted as *num* . Different locations are identified using non-negative integers, denoted as *loc*. Again, *sort* can be any object type listed in Appendix A.1. The properties (*prop*) of an object can include the object type, location, color and size as defined in the following:

$$\begin{aligned}
\mathit{color} &:= \text{white} \mid \text{red} \mid \text{green} \mid \text{yellow} \mid \text{blue} \mid \text{black} \\
\mathit{size} &:= \text{big} \mid \text{small} \\
\mathit{prop} &:= \mathit{sort}(\mathit{num}). \mid \mathit{color}(\mathit{num}). \mid \mathit{size}(\mathit{num}). \mid \mathit{location}(\mathit{num}, \mathit{loc}).
\end{aligned}$$

where $\mathit{sort}(\mathit{num})$. states that object *numś* type is *sort*. $\mathit{color}(\mathit{num})$. states that object *numś* colour is *color*. $\mathit{size}(\mathit{num})$. states that object *num* is of size *size*. $\mathit{location}(\mathit{num}, \mathit{loc})$. states that object *num* is at location *loc*.

Number 0 is the robot, its state *robot* includes its location, the state of the plate and the state of the hand.

$$\mathit{robot} := \mathit{location}(0, \mathit{loc}). \mid \mathit{plate}(\mathit{num}). \mid \mathit{plate}(0). \mid \mathit{hold}(\mathit{num}). \mid \mathit{hold}(0).$$

where $\mathit{location}(0, \mathit{loc})$. states that the robot is at location *loc*. $\mathit{plate}(\mathit{num})$. states that object *num* is on the plate. $\mathit{plate}(0)$. states that the robot plate is empty. $\mathit{hold}(\mathit{num})$. says that the robot hand is currently holding object *num*. $\mathit{hold}(0)$. states that there is currently no object in the robot hand and it is empty.

A statement in the domain description, denoted as *state*, describes either the properties of an object or the state of the robot, defined as below:

$$\mathit{state} := \mathit{prop} \mid \mathit{robot}$$

The recorded domain description in the competition is a set of strings representing *state*. For example,

location(0, 0). plate(0). hold(0). human(1). location(1, 1). big(1). table(2). location(2, 2). big(2). bottle(3). green(3). small(3). location(3, 2).

4.2.2 Task Specification Format

In the command-based league, the file recording the task specification is a set of Instruction Strings (*ins*). For example,

> on(white bottle, worktable). give(human, red bottle). puton(blue bottle, desk). not not on(bottle, table). not goto(teapoy).

In the natural language league, the task specification file contains a set of English sentences that uses the specified vocabulary.

> The white bottle is on the worktable.
> Give me the red bottle.
> Put the blue bottle on desk.
> There must be a bottle on the table.
> Do not go to the teapoy.

4.3 Platform Debugging and Testing

ChallengeServer are provided in both `Debug` and `Release` version. During the development, the following steps need to be taken:

1. Set the project in `Debug` mdoe, and generate the dynamic link library in the `Debug` mode.
2. Copy the generated `dll` in the `$(dir)` and create a `bundles.list` file under the `$(dir)` directory. The file should list all filenames of the dll files (not including the file extension).
3. Run the `Debug` version of the platform. The command line arguments are explained below:
   `cserver.exe [-db dir] [-td dir] [-mode it|nt] [-test id] [-help | -?]`

```
    -bd dir:  set ‘dir’ as directory of bundles.list
    -td dir:  set ‘dir’ as directory of test.list
    -mode it|nt:  set challenge mode, ‘it’ or ‘nt’
      ‘it’ means task is given by instruction
      ‘nt’ means task is given by natural language
    -test id run ‘id’th test
      ‘id’=‘all’ run all the tests
```

During the competition, the competing program needs to provide the dll files for the release version of the ChallengeServer.

4.4 Simulation Platform

Chapter 5

Competition Schedule

This competition has two leagues, the command-based and the natural language league. Each league contains two stages. Each stage contains a set of the domain description and task specification. The competing program needs to calculate and output an action sequence in the given time. The platform will score the competing program according to the action sequenceś performance, and aggregate the scores obtained for each question for ranking.

In the first stage, the task description only contain task/goal specification. Top 8 teams will be allowed to complete in the second stage, where the task specification may contain constraints and extra information. The teams are ranked according to the final total score to select the champion, the runner-up and the third place.

5.1 First Stage

Only the task goal will be described in the task specification. Hereby we introduce a typical question that might occur in the first stage.

Consider the environment setting shown in Figure 5.1. Table 5.1 listed the robot initial state. Table 5.2 provides the properties of large objects. Table 5.3 specifies the properties of small objects. Table 5.4 gives out the in such environment settings, a set of possible task description and planning goal.

Table 5.1: Robot Initial State

location	plate	hand
0	empty	empty

Figure 5.1: Environment Setting Example

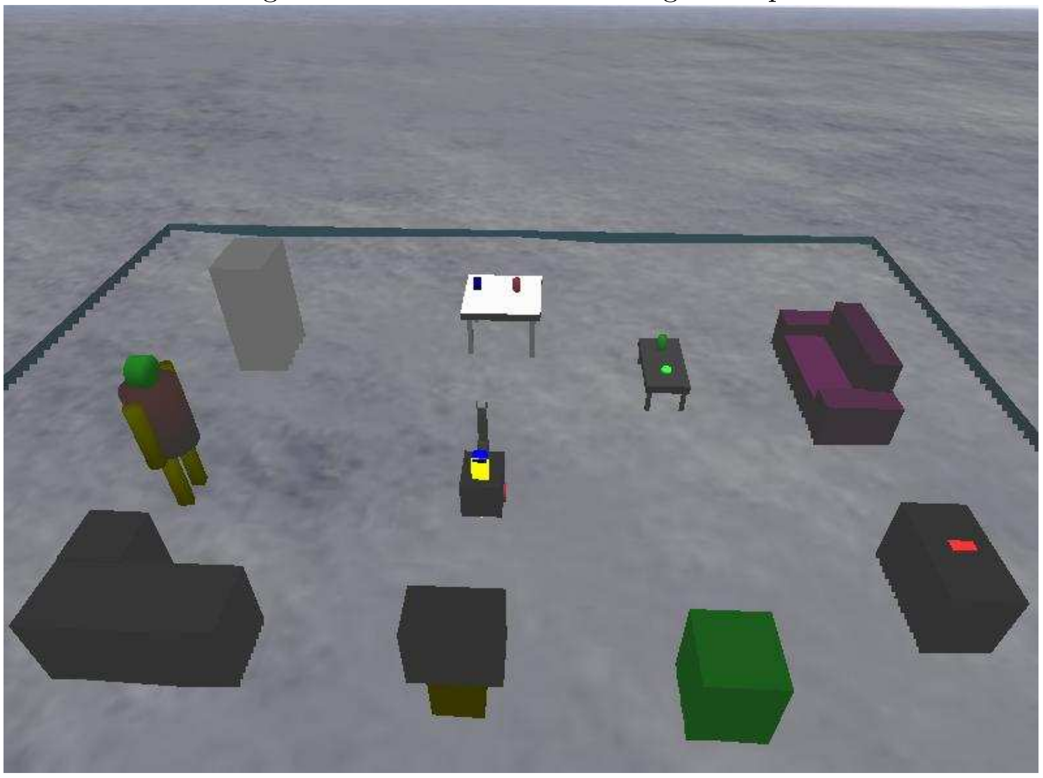


Table 5.2: Description of Large Objects

Object ID	Type (sort)	Colour(color)	Size(size)	Location (loc)
1	human	–	big	1
2	couch	–	big	2
3	table	–	big	3
4	cupboard	–	big	4
5	teapoy	–	big	5
6	television	–	big	6
7	worktable	–	big	7
8	refrigerator	–	big	8
9	desk	–	big	9

Table 5.3: Description of Small Objects

Object ID	Type (sort)	Colour(color)	Size(size)	Location (loc)
10	book	red	small	7
11	can	red	small	4
12	can	green	small	5
13	bottle	red	small	3
14	bottle	blue	small	3
15	bottle	green	small	5
16	remote control	–	small	2
17	cup	blue	small	9

Table 5.4: Task Specification and Planning Outcome

Natural Language Description	Command Instructions	Planning Outcome (not unique)
Give me the green bottle.	give(human, green bottle).	move(5), catch(15), move(1), putdown(15).
Put the red bottle on the teapoy. Put the red can on the teapoy.	puton(red bottle, teapoy). puton(red can, teapoy).	move(3), catch(13), toplate(13), move(4), catch(11), move(5), putdown(11), fromplate(13), putdown(11).
Give me a bottle. Go to the television. Put the cup on the teapoy. Catch a can.	give(human, bottle). goto(television). puton(cup, teapoy). catch(can).	move(9), catch(17), move(5), putdown(17), catch(12), toplate(12), catch(15), move(1), putdown(15), move(6), fromplate(12).

5.2 The Second Stage

The Second Stage often involves constraints and extra information. Based on the above example in the first stage, letś have a look at a typical question in the second stage.

In addition to the description in the first stage, Table 5.5 will be added to provide extra information for the small object. Table 5.6 listed a set of new task sepecifications and expected planning output in the new updated domain. It can be observed that, constraints and extra information significantly increase the complexity of the planning problem.

Table 5.5: Extra Information on Small objects

Object ID	Type (sort)	Colour(color)	Size(size)	Location (loc)
18	book	black	small	–
19	can	blue	small	–
20	bottle	white	small	–
21	bottle	black	small	–

Table 5.6: Task Description and Planning Outcome

Natural Language Description	Command Instructions	Planning Outcome (not unique)
Give me the black bottle. The black bottle is near the blue can. The blue can is on the desk.	give(human, black bottle). near(black bottle, blue can). on(blue can, desk).	move(9), catch(21), move(1), putdown(21).
The white bottle is on the worktable. Give me the red bottle. Put the blue bottle on desk. There must be a bottle on the table. Do not go to the teapoy.	on(white bottle, worktable). give(human, red bottle). puton(blue bottle, desk). not not on(bottle, table). not goto(teapoy).	move(7), catch(20), move(3), putdown(20), catch(13), toplate(3), catch(14), move(9), putdown(14), move(1), fromplate(13), putdown(13).

Chapter 6

The Objectives of the Challenge

Through simulation competition, this challenge aims to stimulate the research on intelligent robotś problem solving, and in particular, intelligent service robotś hight level communication and interaction with human user (including human-robot interaction, automated reasoning and planning). This is to ensure such research can be carried out independent of hardware development, which helps the research and development of service robots (in particular RoboCuphome). Although the competition has deliberately simplified the problem domain, intelligent robot problem solving are richer than the traditional AI planning problems. Mainly, human-robot interaction and reasoning, including natural language based human-robot interaction.

Possible future extension may include

- Providing the Challenge Platform in Open Source.
- Extending the expressiveness of the natural language vocabulary. For example, introducing other pronouns so that complex pronoun resolution becomes necessary.
- Allowing the user to specify domain knowledge or rules, not just facts.
- Extending on the planning part:
  - consider a more complex and larger set of atomic actions.
  - introduce uncertainly in action and perception. For example, the action may be un-deterministic and the action may fail.
  - introduce dynamics into the environment, allow for parallel actions and external interruption. For example, allowing other agents

    in the same environment, updating the domain descriptions during the task completion.

  - allow for quantitative constraints and goal specification, to simulate quantitative decision making that makes use of optimisation.

- Adding functionalities such as question answering system. The user will not only specify the task, but also ask questions so that the robot can answer according to its knowledge base. Meanwhile, allow the robot to obtain knowledge directly from the Internet.

- Adding abilities for automatic questioning and knowledge acquisition. The domain description and user information may be incomplete, and the robot needs to pro-actively ask and obtain knowledge automatically in order to complete the task.

Appendix A

A.1 List of Object Types

The object types allowed in the environment include:

> agent(i.e. the Robot), human (the only user allowed in the environment), couch, cupboard, chair, table, teapoy, book, can, remote control, refrigerator, television, bottle, plant, cup, sofa, air conditioner, workspace, worktable, bed, desk.

Appendix B

Formal Description of the Competition Questions

B.1 Examples of Domain Description in PDDL

B.2 Description of the Atomic Actions in PDDL