EU FP7 CogX
ICT-215181
May 1 2008 (52months)

DR 4.4:
Planning for active learning of new actions and concepts

Moritz Göbelbecker

University of Freiburg
⟨goebelbe@informatik.uni-freiburg.de⟩

Due date of deliverable:	May 31 2012
Actual submission date:	June 30 2012
Lead partner:	ALU
Revision:	final
Dissemination level:	PU

The use of background knowledge can greatly improve the performance of a robotic system, but is usually provided as static knowledge by a system designer, or automatically generated by processes distinct from the system's usual operation. This document describes a software prototype that uses a planner that integrates active learning of new concepts into its usual planning process, allowing it to extend its background knowledge.

Executive Summary

This deliverable presents the work done on creating a planning system that can plan to augment its own background knowledge by using a number of knowledge sources such as human knowledge as well as the environment itself.

We evaluate the software using a set of scenarios in which the planner is tasked with determining a piece of background knowledge, providing it with different means to do so.

This work builds on the systems described in DR.4.3, which can *exploit* background knowledge to plan and act in uncertain environments and DR.7.3 which introduces the ability to *explain* execution failures by assuming that some knowledge may be incomplete or wrong. The system described here goes one step further by being able to *extend* its knowledge and verify whether the previously found explanations were correct.

Role in CogX

One of the objectives of the CogX project is to build a cognitive system that can "self-extend", i.e. autonomously gather new information, extending its knowledge base.

Contribution to the CogX scenarios and prototypes

The work in this deliverable contributes directly to the Dora demonstrator. The previous work on using background knowledge to facilitate goal-directed exploration (e.g. using the knowledge that cereal boxes are commonly found in kitchens to restrict search to rooms that are likely to be kitchens) and on finding explanations for failures were used by the Dora system. The software presented here continues to use the Dora scenario as its main use case and is tightly integrated into the platform.

1 Installation and usage instructions

1.1 Installation

The prototype is delivered in a Ubuntu VirtualBox virtual machine (VM).

- Extract the VM image (either by using an archive manager or the command `tar xvzf dr44.tar.gz`, uncompressed size is 4.2GB)
- Start VirtualBox
- Add the VM to VirtualBox (*Machine→Add* and select the file `Ubuntu CogX DR.4.4/Ubuntu CogX DR.4.4.vbox`)

- Start the VM.

1.2 Preparations

Log in as the default user (ubuntu) using the password "123". Open two terminal windows, change into the "dora" directory and start the CAST-Viewer GUI with `output/bin/display-server&`.

1.3 Starting the system

Start the cast server in one of the terminal windows with the `cast-server` command. Then, in the second window, start the client with `cast-client instantiations/CAST-FILE.cast`. A list of .cast files with sample scenarios is provided in section 2. You can view the planning progress in the *planner.tasks* view of the CAST-Viewer. After a few seconds, the resulting plan should show up here. The *planner.state* page can be used to view the state that is visible to the planner.

1.4 Stopping the system

Simply stop `cast-client` from the terminal using the *Ctrl-C* shortcut and wait a few seconds for the program to terminate. `cast-server` can be stopped in the same way, but this is not requires between runs.

1.5 Compilation

The system can be run as it is, but it can be compiled from scratch with the following commands (starting in the `dora` directory):

```
cd build
cmake ../cmake-caches/dora-dr.4.4.txt ..
make
make install
```

2 Scenario Descriptions

We included four sample planning scenarios in the prototype. They were captured from simulated robot runs and can be started by running `cast-client` with the specified cast-file. The associated planning problems in PDDL format can be found in the `dora/subarchitectures/planner.sa/problems` directory.

2.1 Learning existing knowledge

- **cast-file**: `dr4.4-sample1.cast`
- **pddl-file**: `problem-cerealbox-in-office.pddl`
- **pddl goal**: `(kval robot_0__c (dora__inroom cerealbox office))`

The goal of this task is to find out whether cereal boxes can be found in offices. As this knowledge is already present in the initial state, the goal is already satisfied and the planner returns with an empty plan.

2.2 Learning new knowledge 1

- **cast-file**: `dr4.4-sample2.cast`
- **pddl-file**: `problem-cerealbox-in-magazine.pddl`
- **pddl goal**: `(kval robot_0__c (dora__inobject cerealbox magazine office))`

The goal of this task is to find out whether cereal boxes can be found inside magazines in offices. To humans, this is a quite unusual relation that no one has yet included in the default knowledge, so the planner tries to find a plan that results in knowing that information.

In this instance, the planner makes use of *assumptions* [1, 3], establishing the facts that there is a person in this room and that the person is in a specific place (`place_0__b`). A precondition to the `look-for-people` action is that the room has to be fully explored, which is done by a series of `move` and `move_direct` actions. The look action, together with the assumptions, establishes a state where a person's location is known and the planner proceeds with engaging in a dialogue with that person and asking for the needed information directly (`engage`, `ask-for-bk-inobject`).

2.3 Learning new knowledge 2

- **cast-file**: `dr4.4-sample3.cast`
- **pddl-file**: `problem-container-in-office.pddl`
- **pddl goal**: `(kval robot_0__c (dora__inroom container office))`

Here the task is to find out if containers can be commonly found in offices. Here the room is already explored, and the robot has already engaged with a person. Consequently, the only thing left to do is to ask for the information (`ask-for-bk-inroom`).

2.4 Learning new knowledge indirectly

- **cast-file**: `dr4.4-sample4.cast`
- **pddl-file**: `problem-dr44.pddl`
- **pddl goal**: `(kval robot_0__c (dora__inroom container meetingroom))`

Here the task is to find out if containers can be commonly found in meeting rooms. In contrast to the previous tasks, however, the environment is already explored and it is known to the robot that there are no humans present. The environment consists of four rooms: two offices, a corridor and a meeting room, with the robot starting out in one of the offices.

In this case, direct knowledge gathering is impossible, so the planner falls back to *indirect* gathering by searching for containers in rooms that are likely to be meeting rooms. This is done in the planner by combining assumptions with what we call *knowledge actions*. Whereas assumptive actions describe how facts relate to each other, knowledge actions are derived from assumptions and describe how knowledge about one fact may influence knowledge about another fact.

The plan for this task will use three assumptions:

```
(__commit-category-room_2_61-meetingroom robot_0__c)
(__commit-object-existence-default-0 robot_0__c container room_2_61 meetingroom true)
(__commit-sample_object_location-0 robot_0__c visualobject0 container in room_2_61)
```

The first one assumes that room `2_61` is a meeting room and the second one allows us to make the default assumption that a container exists in the room, as no probability `(dora__inroom container meetingroom)` has been specified. Finally, the information that a "container exists in room `2_61`" is turned into a assumption for the location of a specific object (`visualobject0`).

Then follow the physical actions of the plan: It will first move to room `2_61`, create view points for object search and search for the container object (`visualobject0`) in that room.

The result of these actions, together with the assumptions, result in the fact that the robot knows the location of `visualobject0`. The planner then applies two knowledge actions to get to the goal:

```
(__knowledge-inverse-sample_object_location-1 robot_0__c visualobject0 container in room_2_61)
(__knowledge-inverse-obj_in_room-1 robot_0__c container room_2_61 meetingroom)
```

The first action is derived from the `commit-sample_object_location` assumption and allows the planner to infer from knowing the location of *visualobject0* to knowing whether a container exists in room `2_61`. The second one then allows knowledge transfer to the general "containers exist in meeting rooms" case.

These knowledge actions are only heuristic rules and may fail, as actual knowledge inference is performed by components independent from the planner. For this reason, we don't want to allow the application of knowledge actions if their preconditions are already true in the initial state – for in that case the resulting knowledge should have already been inferred. To implement this, we use the *assertion* mechanism by Brenner and Nebel [2], which achieves exactly that.

References

[1] A. Aydemir, M. Göbelbecker, A. Pronobis, K. Sjöö, and P. Jensfelt. Plan-based object search and exploration using semantic spatial knowledge in the real world. In *Proc. of the European Conference on Mobile Robotics (ECMR 2011)*, Örebro, Sweden, sep 2011.

[2] M. Brenner and B. Nebel. Continual planning and acting in dynamic multiagent environments. *JAAMAS*, 19(3):297–331, 2009.

[3] M. Göbelbecker, C. Gretton, and R. Dearden. A switching planner for combined task and observation planning. In *Proceedings of the Twenty-Fifth Conference on Artificial Intelligence (AAAI 2011)*, page NA, 2011.