Proceedings of the 2006 IEEE International Conference on Robotics and Automation
Orlando, Florida - May 2006

Cooking for humanoid robot, a task that needs symbolic and geometric reasonings.

Fabien Gravot
The University of Tokyo
JSPS Fellowship
Tokyo, JAPAN
Email: fabien@jsk.t.u-tokyo.ac.jp

Atsushi Haneda, Kei Okada
and Masayuki Inaba
The University of Tokyo
Tokyo, JAPAN
Email: {haneda, k-okada and inaba}@jsk.t.u-tokyo.ac.jp

***Abstract*— This paper presents a work toward the old dream of the housekeeping robot. One humanoid robot will cooperate with the user to cook simple dishes. The system will combine predefined tasks and dialogues to find a plan in which both robot and user help each other in the kitchen.**

The kitchen problem allows the demonstration of a large variety of actions, and then the necessity to find and to plan those actions. With this problem the task planner can be fully used to enhance the robot reasoning capacity. Furthermore the robot must also use motion planning to have general procedures to cope with the action planned.

We will focus on the planning problems and the interactions of these two planning methods.

I. Introduction

In the last decade, the interest in "humanoid robots" has greatly increased. They have been presented as the robots which can help in daily life. Until now personal robots are synonym for toys. Only professionals can afford to have useful robots. Robots are certainly expensive but also often need a specific environment to work in. If we can expect robot's price to decrease, the modification of the environment is always a problem. The humanoid robots, by their nature, are able to work in any human environment. So it is understandable to see them as future personal robot.

But because they look like human, the expectations are high. If behavior based reactions are sufficient for a pet robot, a reasoning level is necessary when we face humanoid robot.

In fact the robot desired mission will need a high level reasoning. If you ask a person what he wants humanoid robot to do, you will probably have answers like: cleaning, ironing, cooking, dish washing,... Those missions need both high recognition capacity and planning capacity. And for now cannot be achieved automatically in a general environment. But recent works show a great number of specific application in which part of those missions are accomplished [8], [9].

This paper presents a work toward this old dream of the robot house keeper. We will not address the recognition part of the robot but take it into account into our system. We will focus on the planning problem of cooking.

Cooking is one of the main tasks that an ideal humanoid robot has to be able to perform. The main interest of cooking is its variety of actions. The robot has to cut, mix, pour, bake, ... And naturally it has to be able to do classic manipulation task: pick, place, carry, walk, or less classic: open drawer, close fridge, ... This variety is not found in any other house keeping mission. Then cooking is the problem than enhance the use of a symbolic reasoning level to find and coordinate all the possibles actions. For this reason we will use a task planner.

Humanoid robots have received a great effort to receive autonomy. Most of this work is done on the fundamental functions that are walking and moving. Recently manipulation has also been enhanced [5], [1]. But the tasks are often limited to pick and place. Whereas specific application have addressed more complex tasks [8], [9]. We will try to combine the general motion planner techniques and the specific applications to the task planner to give the maximum of autonomy to the robot.

But even then, a robot which can cook anything by itself is not yet possible or will need several years of works before giving any result. We have then chosen an intermediate goal: the robot cooking helper.

We do not attempt to make a robot chef that can make complex dishes. We will use a HRP2 with arms with 7 degrees of freedom and without any taste sensor to help an user to cook.

Our objective is then easier. It is to make a robot that is at least a good recipes' book and can help time to time to accomplish a part of the cooking. We think that such robot can be useful.

Moreover we want an open system that enables to add new functions into the robot to be more autonomous. The system must be general to be reused but also specific into its application. We do not peel a banana, an orange, an apple, ... in the same way. The action can be sometime very dependent of the ingredient. But other action like "to pour one thing into an other" can be reused often. The robot will do its possible but will ask the user to help to do tasks that it cannot do. This is this cooperation that gives to the system its feasibility.

In the first part, we will present the general architecture and then focus on the task planning system that will organize the cooperation. After we will explain how we use motion planning methods and specific applications to realize the elementary actions. We will finish by a presentation of the main point of the interaction system.

0-7803-9505-0/06/$20.00 ©2006 IEEE

II. General architecture

The general architecture of the system is shown in the figure 1. The Recipes data base has a set of recipes describe by high level task. In fact, a recipe can be viewed as a partial ordered and incomplete plan of high task. For instance "MIX {(100g of butter) (100g of flour) (100g of sugar) (3 eggs)} into (preparation)" "POUR (preparation) into (baking dish)" "BAKE (baking dish) *at (180) for (20 minutes)*". The essential parameters are between brackets but they are not clearly defined. The recipe does not tell where to find the ingredient, in which dish we have to mix them.

We can note also that cooking is the art to create new flavors, then used ingredients disappear and new "preparation" appears. But even more, after cutting it, an ingredient is not really the same. We use the task predicate to define the properties of those ingredients, and they change each time an ingredient is used in an action.

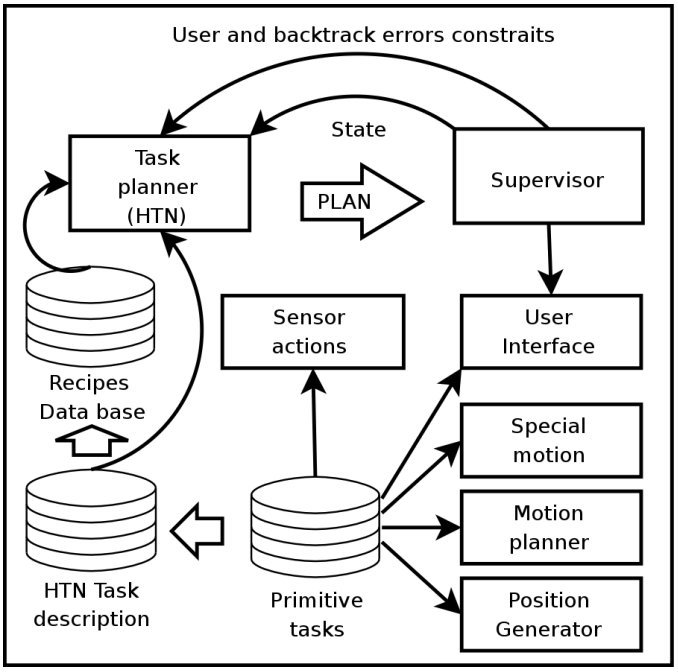

Fig. 1. General architecture of the system.

The Hierarchical Task Network (HTN) [10] describes the methods to achieve the high level task. In fact it describes in a hierarchical way, all the possible choices that has the robot. The robot must use this data base to find a plan of "primitive tasks" representing the high level recipe tasks. For that it will use the task planner.

The primitive tasks have both a symbolic representation and a functional representation. The symbolic representation is used by the task planner and the functional one by the supervisor.

The functional part of the primitive task is defined by a combination of modules of "specific motion", "motion planner", "position generator", "sensor actions" and "user interface".

The supervisor has two fundamental roles. The first is to apply the action planned into the real world. The second is to report the real world modification into the state of the planners and then close the loop.

For its first role, it is in charge of the execution of the primitive tasks by the use of their functional parts.

The second role is to estimate the state of the system. Because we works in simulation it is not a difficult task, but in a real environment it is one of the most challenging aspect of the robotic. It is also to ask the task planner to find a new plan in case of failure of the primitive task. It adds then a constraint to avoid the same choice.

The supervisor ask the task planner for a new plan in three cases:

- Failure backtrack: a primitive task is not applicable du to change in the environment or geometric constraints.
- Change of the *symbolic* state: for instance door opened or closed, but not 18 cm is now 20 cm. It is a change that has a meaning and will influence the symbolic plan. The geometric differences will only influence the motion planners.
- Add of user constraints: The user can always interact with the supervisor to control the robot.

III. Task planning

Our goal is to produce a system able to help the user to make various recipes. A recipe is a plan of high level tasks. The robot must decompose them into sub-tasks to be able to execute them or to see if it must cooperate with the user. Moreover it might exist several possible decomposition methods. The system must be able to find a plan of primitive tasks to realize the recipe.

To represent the possibles actions, we have chosen the Hierarchical Task-Network (HTN). The HTN planning has been studied in Artificial Intelligence for nearly 30 years [10] and continues to challenge more complex problems [6].

After using state of the art planner [7] we have decide to develop our own planner to cope with a more natural way of reasoning on the ingredient properties management.

A. Planning process

To create plans, HTN planners use task decomposition, in which the planning system decomposes tasks into smaller and smaller sub-tasks until reaching primitive tasks that can be performed directly. HTN planning systems have the knowledge of a data base of methods to describe the possible decomposition.

As shown in figure 2, each method includes a description of how to decompose one task into a set of ordered sub-tasks. Moreover there is also restrictions that must be satisfied to apply the method.

Given a task to accomplish, the planner chooses an applicable method, instantiates it to decompose the task into sub-tasks. Then it chooses and instantiates other methods to decompose the sub-tasks even further as illustrated in figure 2. If the pre-conditions of any sub-task are not satisfied, the planning system will backtrack and try other methods.

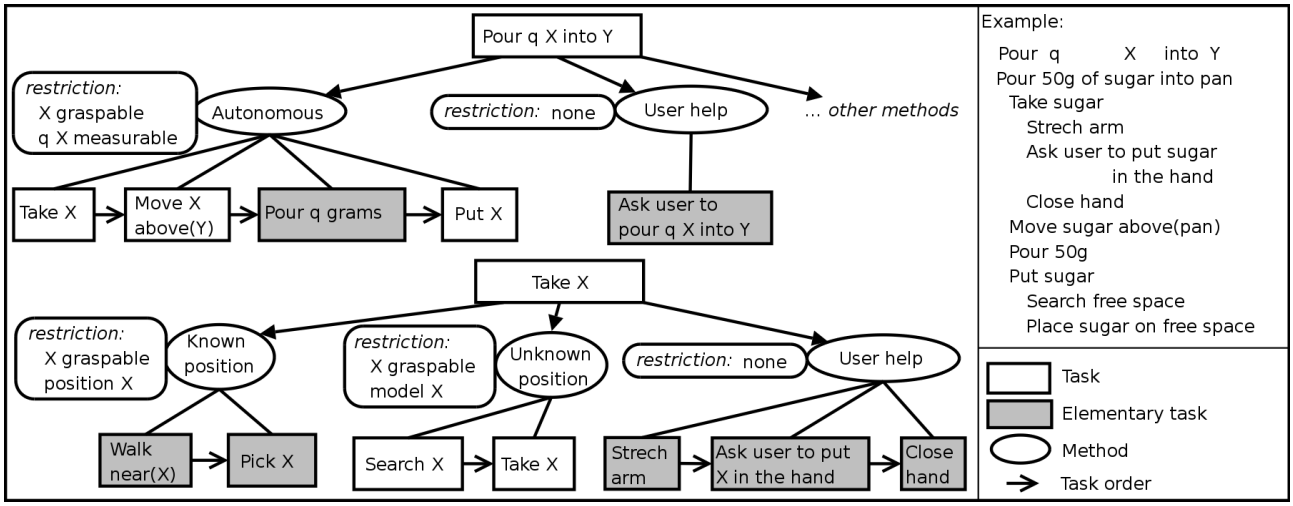

Fig. 2. Example of HTN action description.

This data base of methods represent the knowledge of the system, in this case for cooking. It is given and modified by the programmer. It is the link between all the specific actions and the symbolic reasoning process. If a new action is developed, it can be included into a new method in this data base and increase the choices and then the autonomy of the robot.

At this classic system we have added several modifications:

- External constraints satisfaction: During the test of the pre-conditions of a task we also check if the constraints added by the supervisor are satisfied.
- Partial re-planning: When a re-planning occurs it is rarely between two high level task. We must try to plan with the methods that have been already chosen. And if not use a task to clean the partially executed high level task.
- Method weights: The weights represent the interest of a method. Basically asking the user to do everything is always possible, but not desirable. These weights can increase the tendency to choose autonomous method.
- Partially random choice: When several methods are applicable a random choice is done that take into account the weight of each rule. We do not desire to have a robot that does always the same thing.

Note that the weight system can allow to control how often the robot will try autonomous action. If the user wants to learn a recipe, he cans turn those weights to increase the chance of choosing actions that ask the user help.

B. Primitive tasks.

The system decomposes the recipe into subtasks until it obtains a plan of "primitive tasks". Those tasks are executable by the robot. The primitive tasks will use the four modules to describe applicable actions:

- Motion planner: pick and place, walk, ... (§IV-A).
- Position generator: In this application the symbolic plan is found by the robot, so it has also to find the intermediate positions that are the goals of the motion planner. (§IV-B).
- Specific motion: For instance cutting, mixing, ... These motions are not common in the literature but appears in cooking application. They are characterized by the fact that the trajectory has more meaning that the goal. Then they cannot be planned by a motion planner. They consist in a set of predefined and often repetitive trajectories.
- Sensor action: wait, pour, look for, ... They are associated to a sensor: force sensor for weighting ingredients, camera for searching an object, ... They can also be coupled with motion but there is no planning. For example the figure 2 shows the task "Pour q grams" that inclines the hand until the wrist force sensor feels that q grams have been lost.
- User interaction: dialogue tasks to ask the cooperation of the user (§V).

All primitive tasks have a symbolic description but also correspond to a specific control procedure. The symbolic description allows finding quickly a plan to achieve the recipe. Since it is fast, it is possible to plan a new one if the state changes unexpectedly. In our case the reactivity of the system is given by the task planner time and the current primitive task computation time.

The control procedure of a primitive task is often given by one of the module, but sometime there is also a combination of several modules as we will see in (§IV-C).

When the task is complex and use several modules it is not possible for the task planner to influence the control procedures. Then we can say that the control procedure of a primitive task is given in accordance to the freedom that the robot will have in its choices. The more complex are the control procedures the simpler is the task planning and the more restricted is the robot freedom.

IV. PRIMITIVE MOTION TASKS

Primitive motion tasks are tasks that make the robot moving by combining "motion planner", "position generator", "specific motion" and "sensor action". We will first describe a little the "motion planner" and "position generator" and then give examples of the control procedure of such tasks.

A. Motion planner

The symbolic plan found by the robot gives also the next motion than the robot must perform. But these motions will depend also of the positions of the robot and the objects than are not precisely taken into account in the symbolic level. To have a procedure that can be used for a large scale of objects, we must use motion planning.

Motion planning for a humanoid robot is not a simple task. A lot of works have been done for walking motion [3], [5], but the automatic full body motion planning is yet not so commonly used.

Automatic, full-body motion planning for humanoid robots presents a formidable computational challenge due to:

- The high number of degrees of freedom,
- Motion Planning for Humanoid Robots with complex kinematic and dynamic models,
- Balance constraints that must be carefully maintained in order to prevent the robot from falling down.

We use a version of RRT-Connect that automatically generates collision-free, dynamically-stable motions from full-body posture goals [4]. Obstacle and balance constraints are imposed upon incremental search motions.

The main work that we have done is based on the addition of new constraints during the planning procedure in a way similar to [2]. For cooking we need to manipulate bottle or glasses. When we are carrying them we must keep them at the vertical to not split the liquid. So we must freeze 2 degrees of freedom of the object. We use the inverse kinematic method of Tolani [11] to achieve that.

In the same way, when we open a drawer (figure 5) or a door the hand has only one degree of freedom. This time the hand and the rest of the body follow two different trajectory. The trajectory total is valid if it is possible to close the kinematic chain of the arm for each configuration and if this closure is continuous.

The RRT-Connect planner is the more complex of the 3D planner we use, but not the only one. The RRT-Connect planner is not used for walking. In fact it supposes that the two feet are not moving at all during the trajectory. For walking we have two others motion planner.

A 2D planner that use pre-computed walking procedure. It is the fastest planner and used for long motion.

The step planner can be used for walking in non smooth environment. It uses an elevation map 2.5D. We use it mainly to find the transition between the 2D planner and the RRT-Connect planner.

The description of the planner used is contained into the control procedure of the primitive tasks.

B. Position generator

Some tasks have specific goals, for example to stretch the arm before being given an object in the hand by the user. But generally the goal is to grasp something. Thus we need a procedure to find this goal before calling a motion planner. This is a problem of chain closure of one hand [2].

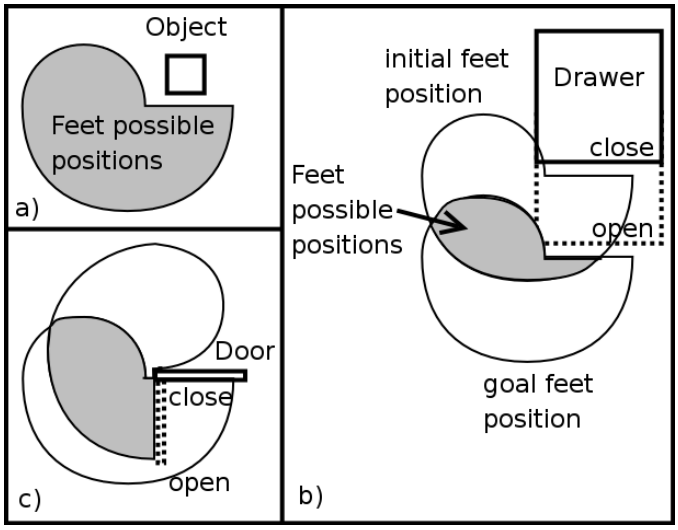

Fig. 3. Zone of randomly chosen feet position when using right arm for a) grasping an object, b) open a drawer, c) open a door.

To solve this problem, we use, in the same way as the planner, a pre-generated set of statically-stable postures [4]. Then we proceed as follow:

1) Select a stable configuration: we select randomly one of pre-generated configurations that has an hand's height similar to the desired grasping goal. When we want to find initial and goal position with the same feet position we select 2 configurations corresponding of the initial and final hand position.
2) Select a position: we choose randomly the robot position (x, y ,theta) without collision and with the hand near the goal position (figure 3). When initial and goal position are needed the zone of feet position is the intersection of the initial zone and goal zone. If we want a given feet position we can go directly to the third step.
3) Gradient descent with collision checks: since the hand position is near the goal we can use gradient descent to try to close the loop. Combined to the gradient descent that is applied on all the degree of freedom, specific inverse kinematics are applied to the legs and arm to precisely close the loop.
4) Check balance: we check the balance of the robot. If it is stable then we have found the goal configuration, else we can try again.

We can note that the gradient descent is not always used since the inverse kinematics can find a solution in one step if the object is near enough to not hate to bend to reach it. This robot use 6 dof legs and 7 dof arms. The inverse kinematics

of the legs have then at maximum 8 solutions but generally only one that satisfied joints limits. The arms however have an infinity of solutions. We use the inverse kinematic method of Tolani [11] that gives the possible intervals of the angle ϕ for an object position and the joints limits. ϕ is the angle of the elbow with the axe shoulder-wrist. It is possible to choose a random value for this angle or the one that is the farther of the joint limits. This selection must just avoid collision.

C. Examples of control procedure

We will see two examples of primitive task control procedure. The first example figure 4 represents the pouring primitive task.

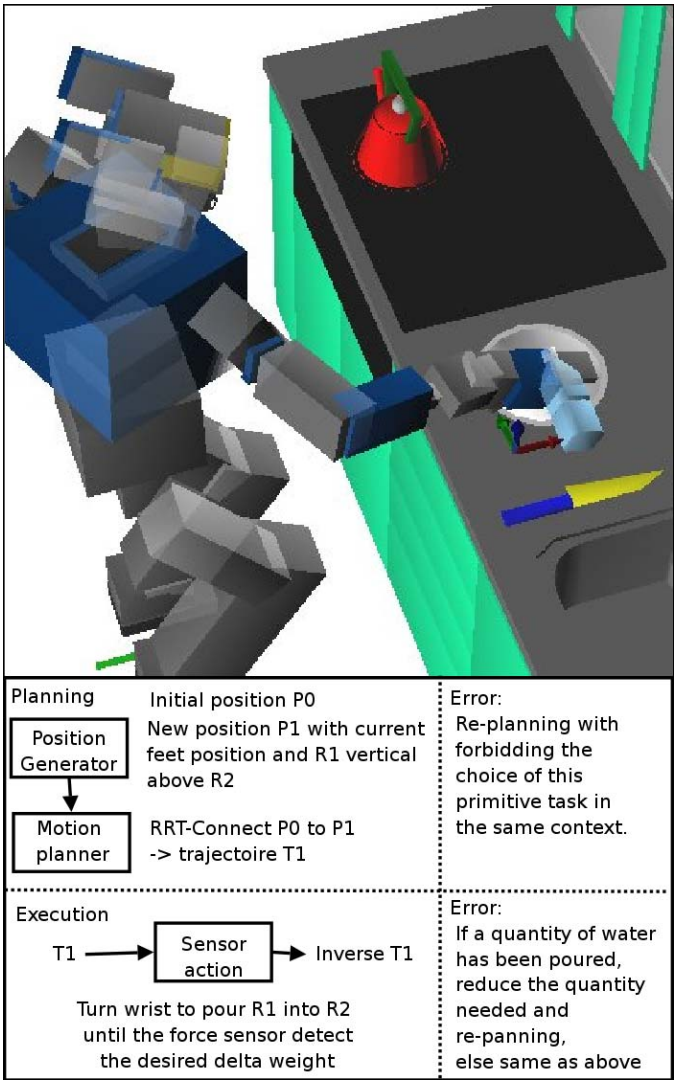

Fig. 4. Example of robot pouring water into a plate.

In precondition this primitive task needs to have a recipient R1 that contain a liquid or a granular ingredient (sugar, rice, ...). The recipient R2 into which the ingredient is poured is known and near.

This means that there are primitive tasks able to satisfy those precondition. For instance ask the user to put R1 in the hand or grasp R1 by itself. Because there are several possibles choices, "grasp" and "pour" are not in the same primitive task.

The control primitive is generally divided into two parts: planning and execution. Sometime there is no planning part. The planning part is only composed of "motion planner" and "position generator" functions. An failure in the planning process always means that the primitive task is not applicable for geometric reason and there is a need to find an other plan without using this primitive task (at least in the same geometric context). So there is re-planning with backtrack error constraint.

The execution part acts on the environment, then the state changes during this process. In case of error we must change the state before re-planning.

An other motion that appears often in the kitchen is to open a door or a drawer. Most of the items are not on the board. Moreover the oven, the fridge are usable autonomously only if the robot is able to open a door (it can also ask the user to do it).

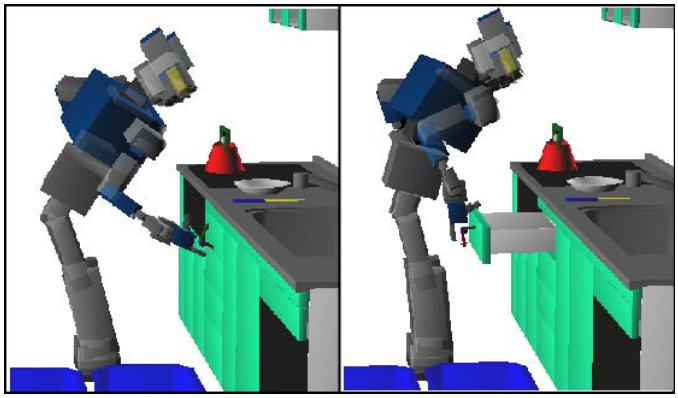

Fig. 5. Full body motion planning for openning a drawer.

This primitive task has only as pre-condition the fact that the drawer is known (position and where to grasp) and to have nothing in its hand. It is not possible to bring the drawer to the robot or to carry the robot. The robot has to move by itself. So this primitive task will be more complex.

In the first part it is similar to the grasping primitive task, decomposition of the motion into 4 phases: the walking phase, the approach, the final motion, and the griper motion. The walking phase use classic 2D walking planner. The approach uses RRT algorithms with balance control. The final motion is a straight line wrist motion. We use inverse kinematics to compute the values of the degrees of freedom. Often this motion has no collision and it is easier to do RRT when we are not too near the obstacles. The last motion is the gripper that grasps the object.

For the drawers (figure 5) we will find two configurations that correspond to the robot with the drawer opened and closed. For this purpose, we use the position generator for initial and goal position (§IV-B). We can then find a common feet position for grasping the drawer in the opened and

closed position. We can then use the RRT-Connect with hand trajectory constraint to find an opening trajectory.

V. Human robot interaction

A. Robot limitations

As we have seen in the previous section, we are able to create autonomous manipulation functions. But there are several limitations. These limits are both software and hardware.

Hardware limitations can be the strength, the speed or the precision of the effectors. If it is easy to imagine a robot cutting jelly; meat or carrot can be more challenging.

Software limitations are programs that have not been yet developed. These will become the next primitive tasks added in the task planner data-base.

Those limitations can be overcome by further research and development or by the use of tips like bar code to identify items. But all limitations cannot be taken into account. We claim that a robot can be useful even if it is not fully autonomous. Then we support the idea of a robot that helps but which can also ask assistance.

For instance, if a robot did not know where is the sugar and is unable to tell the difference between sugar and salt. It is reasonable (for the peace of our stomachs) to think that it can ask the user to give it to him. A robot must seek assistance when it cannot solve a problem by itself.

Since robotic is a very active research field, the system must be able to add new features and characteristics. The general pattern of the HTN representation allows that.

B. Interaction

The system is build with all high level tasks having a method to ask the user help. Because of the weighting strategies of the methods, the robot will search a plan that avoid to use excessively the cooperation of the user. But in case of failure of all autonomous procedures, the system has alway the possibility to only tell the user the recipe. The robot is then reduced to a cooking book.

The interaction procedure is still very rough. There is mainly two parts, the recipe choice and the cooking interaction. The recipe choice can use all the capacity of data-base system (name, theme, ingredients, diet, ...). We will not present this here.

The cooking interaction is still a bit directive. The robot explain all its actions and the actions that it expects from the user and wait for confirmation for the signals of the user when he finished a task. The user can also stop the robot and ask it a new plan by forbidding tasks or adding new actions.

For instance, if the robot is looking for the sugar, the user override this by a command “Do not look for sugar”. The robot will then automatically search a new method and probably ask the user to give it directly the sugar.

An other example is a failure case, when a robot pick the sugar box in a wrong position, the user can correct this by adding a new action “Put sugar” to retry. Because the recipe is only a sequence of partially ordered actions, new action can be inserted in the recipe that will be refined into primitive tasks. In that case “pour 50g of sugar in the pan then ...” will become “put the sugar then pour 50g of sugar in the pan then ...”.

So it is possible to add high level tasks. In fact this function is often used for debugging purpose. The robot ask the user to do some tasks, but the opposite is also true. In this interaction there is always a prior planning to propose adapted answer.

VI. Conclusion and future works.

We have presented a system of a robot that helps for cooking both as a guide and as a assistant. The general pattern used for the symbolic planning system allows refining and expending this capacities. In this article we have presented only the programmer view. The system cannot learn from a non expert. Then it can be useful to add to the system learning capacities.

We have also presented motion planning procedures to autonomously solve several tasks that may need assistance. But the number of specific cooking function is still huge. So we have proposed an alternative to combine the strengths and the weakness of the user and the robot. The robot is precise (recipe, weight, time) and do not get bored. The human can taste, learn, recognize nearly everything.

Most of this work has been done in simulation and the next months will probably see more experiments and the definition of new tasks. The main obstacle will be to have a good “supervisor” to modify the world representation used here with the real world modifications in both a symbolic and geometric level.

References

[1] A. Sahbani and J. Cortes and T. Simeon. *A Probabilistic Algorithm for Manipulation Planning under Continuous Grasps and Placements* IEEE International Conference on Intelligent Robots and Systems, 2002

[2] J. Cortes and T. Simeon and J.P. Laumond. *A Random Loop Generator for Planning the Motions of Closed Kinematic Chains using PRM Methods.* IEEE International Conference on Robotics and Automation, 2002

[3] K. Hirai. *Current and future perspective of Honda humanoid robot.* Proc. IEEE/RSJ Int. Conf. Intell. Robot. & Sys. (IROS), 1997

[4] J.J. Kuffner, S.Kagami, K. Nishiwaki, M. Inaba, and H. Inoue. *Dynamically-stable motion planning for humanoid robots.* Autonomous Robots (special issue on Humanoid Robotics), 12, 2002.

[5] J.J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, H. Inoue. *Motion Planning for Humanoid Robots* Proc. 11th Int. Symp. of Robotics Research (ISRR 2003).

[6] U. Kuter & D. Nau. *Forward-chaining planning in nondeterministic domains.* Proceedings of AAAI, 2004

[7] D. Nau, H. Munoz-Avila, Y. Cao, A. Lotem, and S. Mitchell. *Total-Order Planning with Partially Ordered Subtasks.* In IJCAI-2001. Seattle, August, 2001

[8] K. Okada, M. Hayashi and M. Inaba, *Development of Waterproof Glove for Humanoid Robots*, ROBOMEC, Kobe, Japan, 2005

[9] K. Okada, T. Ogura, A. Haneda, J. Fujimoto, F. Gravot and M. Inaba, *Humanoid Motion Generation System on HRP2-JSK for Daily Life Environment*, International Conference on Mechatronics and Automation (ICMA05), 2005

[10] E. D. Sacerdoti. *A Structure for Plans and Behavior.* American Elsevier Publishing Company. E. D. 1977.

[11] D. Tolani, A. Goswami and N. I. Badler. *Real-Time Inverse Kinematics Techniques for Anthropomorphic Limbs* Graphical Models 62, 2000