Cognitive Robotics
AAAI Technical Report WS-12-06

A Robust Planning Framework for Cognitive Robots

Sertac Karapinar and **Dogan Altan** and **Sanem Sariel-Talay**
Artificial Intelligence and Robotics Laboratory
Computer Engineering Department
Istanbul Technical University, Istanbul, Turkey
{karapinarse,daltan,sariel}@itu.edu.tr

Abstract

A cognitive robot should construct a plan to attain its goals. While it executes the actions in its plan, it may face several failures due to both internal and external issues. We present a taxonomy to classify these failures that may be encountered during the execution of cognitive tasks. The taxonomy presents a wide range of failure types. To recover from most of these failures presented in this taxonomy, we propose a *Robust Planning Framework* for cognitive robots. Our framework combines planning, reasoning and learning procedures into each other for robust execution of cognitive tasks. Failures can be detected and handled by reasoning and replanning, respectively. The framework also facilitates learning new hypotheses incrementally based on experience. It can successfully detect and recover from temporary failures on a selected set of actions executed by a Pioneer3DX robot. It has been shown that our preliminary results for hypothesis learning in failure scenarios are promising.

Introduction

A cognitive robot should possess abilities to solve problems and plan to attain its goals, reason about dynamic cases and learn from experience as intelligent systems in nature. Problem solving and planning is crucial for achieving the given objectives. Automated planners are commonly used for finding a coarse of actions for a robot to achieve its goals. These planners usually take the domain information (initial/goal states and operators corresponding to real-world actions) to construct a plan. During the execution of actions in the constructed plan, a robot may face several types of failures some of which may be recovered by replanning. However, there may be gaps between the real-world representation of the domain and its symbolic counterpart. Especially when the real outcomes of actions are not completely represented, a planner may not be able to construct a valid plan in case of failures. Belief revision and reasoning tools are necessary to deal with these type of issues. Furthermore, the robot should be equipped with learning capabilities for the efficiency of its future decisions.

Our research focuses on developing a robust planning framework against real-world failures. The framework integrates planning/replanning, generic belief revision, reasoning and learning processes together to enable robust execution of tasks that require cognition. We investigate several different types of failures that may occur during the execution of a plan and propose a taxonomy for robot failures in cognitive tasks.

Detecting failures and recovering from them are studied in a previous work (Jonge, Roos, and Witteveen 2009). Some existing methods address failure handling through repairing plans (van der Krogt and de Weerdt 2005; Micalizio 2009; Gianni et al. 2011). A classification of failures is given in a previous work to detect inconsistencies between the theory and the model of the real world, plan execution and actual observations (Steinbauer and Wotawa 2009). Metacognitive loop (MCL) has been proposed to deal with failures with a graph structure that represent solutions to failures (Schmill et al. 2007). A recent work (Hermans, Rehg, and Bobick 2011) investigates robot learning for predicting the effects of actions. Visual attributes of objects are used for affordance prediction to utilize planning and action selection. The same idea may be applied for failure handling.

Action execution failures have also been addressed in planning frameworks (Gobelbecker et al. 2010). Some explanations for failures may be found and the robot may be forced to replan with the help of excuses. Excuses could be extracted by changing the initial state such that a valid plan can be constructed from there on. Beetz et al. (2010) propose a system to cope with plan execution flaws. Their system involves plan projection and transformation processes in order to detect behavior flaws. Plan projection is achieved by simulating the plan execution in a physics-based simulator. In case of a behavior flaw in the plan projection, transformation rules are applied to find a valid plan.

In our previous study (Usug and Sariel-Talay 2011), failures are handled by considering all actions that can manipulate the cause of the failure. This means that the robot tries to perform each of these actions until it handles the failures and succeeds in achieving the goal. This might lead the robot to execute irrelevant actions, causing degradation in performance.

Our approach differs from earlier work in the way the failures are handled. Domain knowledge, if available, is used to reason about failures and recover from them; otherwise, an intuitive approach could be applied to construct alternative plans (Usug and Sariel-Talay 2011). We use TLPlan tempo-

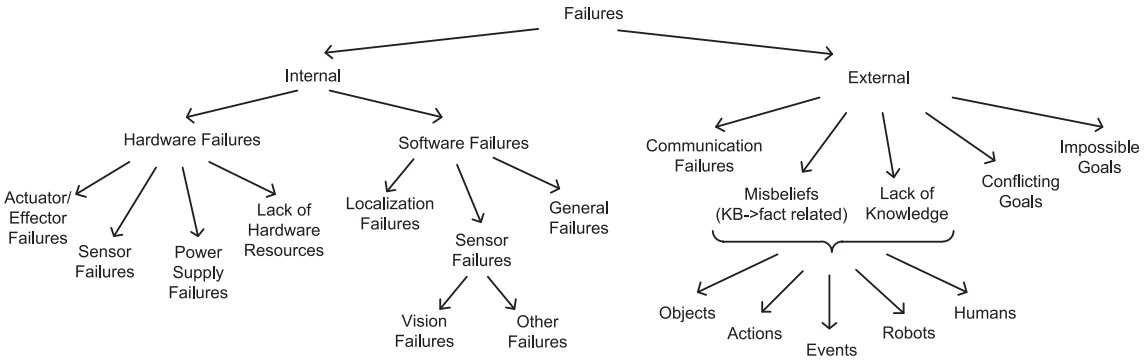

Figure 1: A Taxonomy of Robot Failures in Cognitive Tasks.

ral planner to plan and replan when failures are detected. ProbCog reasoning tool (ProbCog 2012) is used to reason about failures. An Inductive Logic Programming (ILP) based approach is used for belief revision for further decisions.

This paper is organized as follows. First, we present the overall problem of the planning task for a cognitive robot and a taxonomy of robot failures for cognitive tasks. Then, we describe the details of our proposed planning framework to solve the stated problem. The preliminary results of our framework are presented in the following section. Finally, we conclude the paper with discussions and future work.

Robust Planning for Cognitive Robots

A cognitive robot is expected to solve a complex cognitive problem that is usually presented as a planning task. The robot takes the given planning problem and domain knowledge to solve its problem and attain its goals by using an automated planner. The planner generates a sequence of actions as a plan. While executing its planned actions, the robot may encounter several failures. The robot can handle a failure by generating a new plan. However, in some cases replanning does not adequately solve the problem. The robot should reason about these cases and learn from experience.

We formulate the planning task for a cognitive robot, then present a taxonomy of different types of failures that may occur during the execution of a plan. Finally, we state the main robust planning and execution problem that we would like to solve.

Planning Task for a Cognitive Robot A planning task is represented by a tuple $\Pi = (\Delta, s_0, s^*)$ where Δ is a planning domain, s_0 is the initial state, and s^* is the goal state. Planning domain is a tuple $\Delta = (T, C, S, O)$ where T is the set of types, C is the set of domain constants, S is the set of predicates and O are the planning operators.

A planning operator $o \subseteq O$ can be formalized as a triple $o = \{\text{pre}(o), \text{add}(o), \text{del}(o)\}$. Planning task is achieved by a planner to reach a goal state s^* from the initial state s_0 by applying the selected operators O at consecutive states in the given order. An operator o can be selected by the planner at a state $s \subseteq S$ only if $\text{pre}(o) \subseteq s$. After applying o at state s, a new state $s' = \text{add}(o) \cup (s \setminus \text{del}(o))$ is observed.

The robot should maintain a knowledge base $KB = (F, R)$ where F is the set of facts and R is the set of rules. The aim of the robot is to make new conclusions if $KB \models \alpha$ where α is a new query and update the KB according to this inference result.

If all goes well for the planning task, the robot successfully attains its goal. However, since robotic environments are highly dynamic and uncertain, the robot may not succeed in executing its plan due to different types of failures. The following subsection presents our taxonomy of failures that may occur during runtime.

A Taxonomy of Robot Failures in Cognitive Tasks We propose a taxonomy of robot failures in cognitive tasks (Figure 1). The taxonomy presents two main categories, namely, external and internal failures. External failures are related to environmental issues whereas internal failures are more about the robot's internal state. Internal failures are divided into two subcategories as hardware and software failures. Hardware issues are about component failures of the robot. This branch includes five sub-branches: (1) Actuator/effector failures (2) Sensor failures (3) Power supply failures (4) Lack of hardware resources. The actuators or the effectors of the robot may fail due to mechanical or electrical issues. Sensor failures may represent a state where sensory information is partially or completely lost. A robot may also fail to perform any tasks due to its dead battery. When the robot lacks the required hardware resources to execute an action, it may fail to achieve it. Assume that the robot is assigned a *pick up* action. If the object is too heavy for the robot to lift, the robot may fail to achieve this task. The other branch of internal failures -software failures- include three main categories, namely, localization, sensor and general failures. Localization failures are about errors in localization. While sensor failures are about incomplete sensor model, general failures are about other failures which occur due to the other software issues of the robot.

The second main category of failures includes external failures. A similar classification for external failures is given in an earlier work (Beetz 2000). Our taxonomy presents five sub-branches for external failures: (1) Communication failures (2) Misbeliefs on the facts and the rules in the KB (3) Lack of knowledge (4) Conflicting goals (5) Impossi-

ble goals. Communication failures occur when the robot is not able to communicate with other robots or with an operation center in the environment. The same instance may occur when robots collaborate with each other through communication. The robot may also have misbeliefs in its KB of the pre/post conditions of actions or events. That may result in failures in achieving tasks. Action-related failures may occur due to incomplete representation of actions in the KB. For instance, actions may lead to unexpected outcomes in the environment. Similarly, the robot may not be aware of some events and their effects in the environment. Object-related failures arise from false beliefs about an object's existence, location or its attributes. These types of failures may arise from visual features (e.g., SIFT, VPFH etc.) as well as the mobility-related features such as weight, grasp position and center of mass (COM). For example, when the robot tries to *pick up* an object, it may detect that the object is wider than the size of its gripper. However, if the robot detects that the object has a handler, it may recover from the failure by grasping it from its handler. Misbeliefs about other robots may cause timing problems and this leads to failures especially in cooperation. In such a case, when the other robot delays or somehow fails, the robot may probably fail. The robot may also fail if it has misbeliefs about humans in the environment. When a domestic service robot has erroneous preference model for its companion, the robot may fail in achieving the desired goal. Lack of knowledge about objects, actions and events may also result in failures. Robots may also fail in achieving their tasks when their goals conflict with each other. Finally, when the robot detects that its goal is impossible to achieve, there is no way to attain the given objective. For instance, if the goal of the robot is going outside of a room without a door, the robot will not be able to reach its goal under this circumstance.

Note that failures in path planning are not presented in this taxonomy since we address failures in higher level cognitive tasks. Although there seems that failure types are distinct from each other, simultaneous failure causes or chain of events may result in failures.

These failure modes can also be classified under two main categories: complete and partial failures. Complete failures model the situations where the robot is not able to execute any actions. Assume that the software system of the robot completely crashes. In this case, the robot may not even be able to find out the reason of the failure. On the other hand, whenever a partial failure occur (e.g., a sensor failure) the robot may find a way to handle the failure.

Robustness in Planning and Execution During the execution of a plan, the robot may face different failure situations presented in our taxonomy (Figure 1). After detecting a failure, it should recover from the failure. Considering their recovery processes, the failure types can be grouped under two categories, namely, *temporary* and *permanent failures*.

Definition 1 (Temporary failures). A temporary failure occurs during the execution of a plan. These types of failures can be resolved by replanning. Temporarily failed actions may become available whenever replanning is possible.

Definition 2 (Permanent failures). A permanent failure cannot be resolved by replanning. Therefore, there is no way to execute a permanently failed action.

The problem that we investigate asks for finding the reasons of failures during the execution of a plan and recovering from them if they are *temporary failures*. Therefore, the robot should be able to determine the type of a failure as whether *temporary* or *permanent*. A belief revision and learning mechanisms are needed for this purpose. Our main objective is to handle failures that are included in the category of external failures for single robot cases, partial hardware failures and lack of hardware resources.

A Running Example

As a motivating example to illustrate the above mentioned problem, a scenario can be given with several objects to be moved from their starting positions to the given goal positions. The robot is able to execute several actions such as *pick up/put down* objects by its gripper, *move* from one location to another. However, the robot might drop an object while trying to pick it up. This failure might occur because of several reasons such as the weight of the object or wrong grasp position. The robot can handle this *temporary failure* by executing *pick up* action from different orientations if the reason of the failure is its wrong grasp position. This type of failure occurs due to a "misbelief" on the *pick up* action for the related object. If the robot fails in executing *pick up* action because of its weight, there is no way to achieve this action due to a *permanent failure*. Replanning may lead the robot keep dropping the object repeatedly. In this case, the cause of the failure is "lack of a hardware resource". On the other hand, another action (e.g., *push*) which does not directly contribute to the robot's goal can handle this type of failure. To resolve the failure, this action should be added into the plan. A reasoning mechanism can figure out the cause of the failure in such cases and updates the KB so that the necessary actions are included in the plan (Usug and Sariel-Talay 2011).

Robust Planning Framework

We propose a robust planning framework for cognitive robots. The framework combines six modules, namely, Planning module, Scene/Object understanding module, Execution module, Execution & Plan Monitoring module, Geometric Reasoning module and Learning module, all of which are connected to a Knowledge Base (KB). KB maintains the *domain knowledge*, the world state and goals (i.e., planning problem), the *plan* and the gained *experience*. The modules that embody the framework and information flow among them are illustrated in Figure 2. These modules are to be connected to sensor and motor interfaces of a robot system. Note that mapping and path planning modules are not illustrated in the figure for clarity.

Planning module constructs high-level plans for the given goals. It uses the domain knowledge (i.e., operators and facts) and the current world state to generate a valid plan. TLPlan (Bacchus and Ady 2001), a forward chaining temporal planner, is used as the core of the *Planning module*. It can construct plans for durative and concurrent single/multi

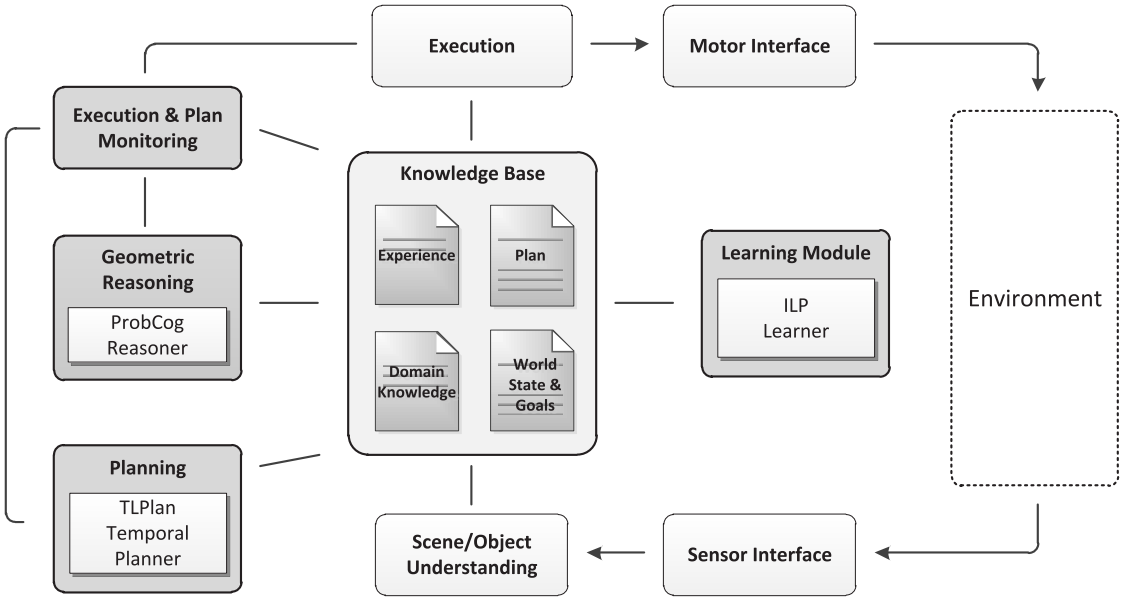

Figure 2: Robust Planning Framework for Cognitive Robots.

robot actions. When a complete plan for a given state is found, it is maintained in the KB so that the *Execution module* can execute the actions in the order.

Scene/Object understanding module is responsible for updating the domain knowledge in KB based on the gathered data from the environment. *Execution module* executes each action in the plan by considering the incoming perceptual information. This module is connected to the actuators and the effectors of the robot. *Execution & Plan Monitoring module* gets information from the *Execution module*, monitors the execution of the plan and detects failures if any. Upon detecting failures, it invokes the planner to force it replan for the updated world knowledge.

Geometric Reasoning module reasons about the failure and alters the parameters of the failed actions when needed. This module uses ProbCog as a first-order probabilistic reasoner (ProbCog 2012). The observed data are processed by the reasoner and the KB is updated based on the results of the reasoning process so that *Execution module* uses this information.

Learning module is responsible for the robot to adapt itself based on the gained *experience* about actions, their parameters and everything in the environment affected by these actions.

Detecting Failures

Detecting failures is crucial for efficiently recovering from them. Similar to human-level failure detection (Wolpert and Ghahramani 2000), our framework detects failures by confronting sensory predictor with the actual sensor inputs. For instance, the robot may fail during the execution of a *pick up* action due to several reasons. The *Scene/Object understanding module* uses both camera and tactile sensor data to infer the state of the object in interest and register the state information in the KB. *Execution & Plan Monitoring module* continuously monitors the execution of the plan. If the observed state is different than the intended outcome, a failure is assumed and the corresponding replanning and reasoning methods are activated.

Integrating Reasoning into Planning

The *Geometric Reasoning module* uses the KB and the incoming information from the *Execution & Plan Monitoring module* and updates the KB when new conclusions can be made. Two types of conclusions can be made: (1) the parameters of a failed action are updated to handle failures, (2) a permanent failure is detected when the robot gives up executing an action that fails after several trials. The conclusions are also told the KB as a part of the *planning problem*. The updated *world state* is fed into the *Planning module* to replan for the given situation. In this process, the robot tries to update the low level representation of an action if the *Execution & Plan Monitoring module* detects a failure. The robot insists on executing a failed action by changing its parameters (i.e., different grasp positions or different trajectories). These trials continue until either the action is successfully executed or the robot reaches at the limits of its *patience* (i.e., a patience threshold as in (Parker 1998)). If the threshold is exceeded, the robot stops trying alternatives, and abandons the execution of that action. All the updated actions and the information related to these actions are maintained in the robot's *experience* so that the *Learning module* can either build up a new hypothesis or update an existing one about these actions.

The main procedures for robust planning, execution and geometric reasoning are shown in Algorithms 1-3, respectively. The main algorithm keeps running until the goal state is achieved or no plan can be found. Whenever a valid plan

Algorithm 1 Robust Planning(Δ, s_0, s^*)

Input: Planning domain Δ, initial state s_0, goal state s^*
Output: returns $success$ or $failure$

$S = s_0$
while $s^* \not\subseteq S$ **do**
  P = Planner(Δ, S, s^*)
  if $P = \varnothing$ **then**
    return $failure$
  end if
  S = Execution(S, P)
end while

return $success$

is found, each action in the plan is executed in the order by Algorithm 2 while monitoring the execution at the same time. This algorithm checks whether actions are successfully executed or *permanent failures* are detected by the *GeometricReasoning* algorithm. For both success and failure cases, KB is updated. Based on these updates, hypotheses are constructed or updated by applying the ILP learning method.

GeometricReasoning algorithm generates new execution modes by using the ProbCog reasoner. These execution modes correspond to alternative execution scenarios of a temporarily failed action. Note that a failure is assumed to be a *temporary failure* initially. However, after several trials, if the patience of the robot for the failed action is exceeded, a *permanent failure* assumption is made and the algorithm returns failure. In a *temporary failure* case, the updated action with an alternative execution mode is sent back to the Algorithm 2.

The following update from our example scenario shows how our reasoning module works. The robot tries to pick up a red cubical object. When it fails in its first attempt, a new execution mode is proposed by ProbCog for *pick up* action so that the robot changes its initial orientation (α) with respect to the object and tries to pick up the object again from a different angle (β). If it fails in its second attempt, the orientation is updated again. At the end of the third attempt, the experience includes the following facts:

$$Red(x) \Lambda Cube(x) \Lambda Orientation(\alpha, x) \Lambda PickUpFailed(x)$$
$$Red(x) \Lambda Cube(x) \Lambda Orientation(\beta, x) \Lambda PickUpFailed(x)$$
$$Red(x) \Lambda Cube(x) \Lambda Orientation(\gamma, x) \Lambda PickUpFailed(x)$$

During these trials, if the patience threshold of the robot is exceeded, it stops trying alternative execution modes for *pick up* action on the corresponding object. If the above facts are observed, about *pick up* action, a *permanent failure* assumption is made when the patience threshold is exceeded, and the learning module constructs the following hypothesis:

$$[Red(x)\ \Lambda\ Cube(x)] \Rightarrow PickUpFailed(x)$$

After trying to pick up other red cubical objects and making new observations, this hypothesis might be strengthened or rejected. Note that the patience threshold could be determined based on the action type. We leave a detailed discussion of this parameter as a future work.

Algorithm 2 Execution(S, P)

Input: Current state S, plan P
Output: Current state S

while P ! $= \varnothing$ **do**
  $action$ = POP(P)
  $action.patience = action.threshold$
  while ($execMonitor$ = Execute($action, S$)) = $failure$
  //Temporary failure is assumed
  do
    KB.Add([$action$, $execMonitor$])
    [$action$,$geoResult$] = GeometricReasoning($action$)
    //Permanent failure is assumed
    if $geoResult$!= $success$ **then**
      H = ILP-Learning($KB, action$)
      KB.Update([H, $action$]) //Update the hypothesis
      return S
    end if
  end while
  KB.Add([$action$, $execMonitor$])
  H = ILP-Learning($KB, action$)
  KB.Update([H,$action$]) //Update the hypothesis
end while

return S

Algorithm 3 GeometricReasoning(a_f)

Input: Failed action a_f
Output: updated action a'_f and $success$ or $failure$

$p = a_f.patience$
Decrease($a_f.patience$)
if $p >= 0$ **then**
  return [$a_f, failure$]
end if
a'_f = ProbCog(a_f)

return [$a'_f, success$]

Interleaving Learning with Planning

Our learning method is based on *Inductive Logic Programming (ILP)*. In ILP, the main purpose is finding a hypothesis that is consistent with the observed data. The hypothesis is expressed by a set of logical sentences. ILP tries to take advantage of inductive methods. We have implemented Top-down approach for the robot to learn hypotheses based on evidences.

ILP helps to build, update or abandon hypotheses as the robot acquires new observations. In this framework, hypotheses express relations between failed actions and their parameters.

Top-down Approach The logic-based learning algorithm employed in the *Learning module* is the top-down inductive learning approach. It starts with generating a general hypothesis, then, it tries to narrow down the expression of the hypothesis such that it agrees with the observed data. The

top-down learning algorithm is divided into two parts which are given in Algorithm 4 and Algorithm 5. Inputs of the Algorithm 4 are the *observations* and the *target (i.e., failed action, a_f)*. *Target* is the predicate that we want to construct a hypothesis for, and *observations* are the evidences that the robot collects from the environment. Algorithm 4 repeatedly constructs hypotheses until all positive examples (in our case, all observations that lead to action failures) are covered. In each iteration, Algorithm 5 is invoked which finds the action parameter that leads to action failures most. Naturally, that parameter might also be the parameter of successful actions. Therefore, we need to eliminate negative examples (successful actions). In this step, the algorithm finds the parameter that covers the negative examples most, and appends the negation of this parameter to the previous clause. Algorithm 5 stops when all the negative examples are covered completely.

Algorithm 4 ILP-Learning(O, T)

Input: observations O, target T
Output: hypothesis H

$H \leftarrow NULL$
while O *has positive examples* **do**
  $L \leftarrow ChooseLiterals(O, T)$
  $H \leftarrow H + L$
  $O \leftarrow O -$ *positive examples covered by* L
end while

return H

Algorithm 5 ChooseLiterals(O, T)

Input: observations O, target T
Output: literal L

$L \leftarrow$ *literal that covers the positive examples most*
$O' \leftarrow$ *examples covered by* L
while O' *has negative examples* **do**
  $L' \leftarrow$ *literal that covers the negative examples most and does not appear in the positive examples together with* L
  $L \leftarrow L + \neg L'$
  $O \leftarrow O -$ *negative examples covered by* L'
end while

return L

As an example, suppose we have the following hypothesis at an intermediate step:

$$[Red(x) \wedge Cube(x)] \Rightarrow PickUpFailed(x)$$

which means that whenever the robot tries to pick up red cubical objects, it always drops the object. After collecting some inconsistent evidences, this hypothesis should be updated. If this is the final hypothesis, the robot should avoid executing picking up the red cubical objects at its first sight. However, if the robot observes the following world state:

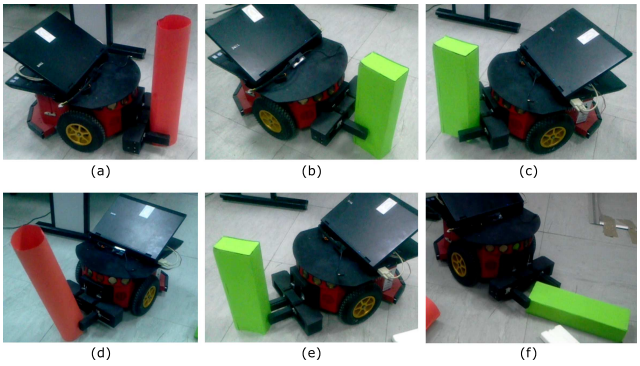

Figure 3: A sample set of successful and failed *pick up* actions by a Pioneer3DX robot. Successful and failed scenarios are presented in the top and bottom rows, respectively.

$$Red(y) \Lambda Cube(y) \Lambda Orientation(\beta, y) \Lambda \neg PickUpFailed(y)$$

then, it has to reject the previous hypothesis. Its future decisions about the *pick up* action for different types of objects is made according to the *Left Hand Side (LHS)* of the hypothesis. When any of the predicates in the LHS are changed, the decision of the robot will also change accordingly.

Preliminary Results

To evaluate the success rate of our learning system, we have used a training set with 30 different *pick up* actions. These training data are used to construct a hypothesis for the failed actions. A sample set of successful and failed *pick up* actions are illustrated in Figure 3. Success and failure scenarios are presented in the top and bottom rows, respectively.

We have considered five different attributes during classification: the *color*, *shape*, *size*, the *orientation* of the object and the *distance* between the robot and the object during the execution of the action. We have compared the *ILP* algorithm with the *Bayes Network classification*. Five different measurements are considered: *true positive rate (TP)*, *false positive rate (FP)*, *precision*, *recall* and *F-score*. The result for *ILP* can be seen in Table 1 where Pick-S and Pick-F are for successful and failed *pick up* actions, respectively. The results for the *Bayes Network classification* are given in Table 2. The hypothesis constructed by *ILP* can correctly classifies 24 of the *pick up* actions out of 30, while the hypothesis by the *Bayes Network classification* can correctly classifies 22 of them. As can be seen from these results, *ILP* gives better results than the *Bayes Network classification*.

TP	FP	Precision	Recall	Class	F-score
0.87	0.27	0.77	0.87	Pick-S	0.81
0.73	0.13	0.85	0.73	Pick-F	0.79

Table 1: The overall results of ILP

TP	FP	Precision	Recall	Class	F-score
0.87	0.40	0.68	0.87	Pick-S	0.76
0.60	0.13	0.82	0.60	Pick-F	0.69

Table 2: The overall results of Bayes Network Classification

Conclusion

In this paper, we investigate the reasons of failures that may occur during the life time of a cognitive robot, and propose a taxonomy for these failures. The ultimate goal is to determine whether an encountered action failure is a *temporary failure* or a *permanent failure* and to recover from this failure. In our framework, planning is continually performed and the necessary precautions are taken to cope with temporary failures. The framework integrates reasoning tools into planning processes to update actions that temporarily fail. The framework also includes ILP learning processes to generalize hypotheses about the failed actions according to the robot's experience. Our results are promising in the sense that hypothesis learning is successful to a great extent. Our future work includes a detailed analysis of the performance of the framework on different scenarios, investigation of the patience thresholds and failure detection procedures for different types of actions.

Acknowledgements

This research is funded by a grant from the Scientific and Technological Research Council of Turkey (TUBITAK), Grant No. 111E-286. TUBITAK's support is gratefully acknowledged. Authors also would like to thank Dr. Hulya Yalcin for helpful discussions and Burak Sarigul for his contributions in the robot experiments.

References

Bacchus, F., and Ady, M. 2001. Planning with resources and concurrency a forward chaining approach. In *Proceedings of the 17th international joint conference on Artificial intelligence - Volume 1*, IJCAI'01, 417–424. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Beetz, M.; Jain, D.; Msenlechner, L.; and Tenorth, M. 2010. Towards performing everyday manipulation activities. *Robotics and Autonomous Systems* 58(9):1085 – 1095.

Beetz, M. 2000. *Concurrent Reactive Plans: Anticipating and Forestalling Execution Failures*. Number 1772. no. in Lecture Notes in Computer Science. Springer.

Gianni, M.; Papadakis, P.; Pirri, F.; Liu, M.; Pomerleau, F.; Colas, F.; Zimmermann, K.; Svoboda, T.; Petricek, T.; Kruijff, G.-J.; Khambhaita, H.; and Zender, H. 2011. A unified framework for planning and execution-monitoring of mobile robots. In *Proceedings of the AAAI-11 Workshop on Automated Action Planning for Autonomous Mobile Robots (PAMR)*. AAAI.

Gobelbecker, M.; Keller, T.; Eyerich, P.; Brenner, M.; and Nebel, B. 2010. Coming up with good excuses: What to do when no plan can be found. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Hermans, T.; Rehg, J. M.; and Bobick, A. 2011. Affordance prediction via learned object attributes. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA): Workshop on Semantic Perception, Mapping, and Exploration*.

Jonge, F.; Roos, N.; and Witteveen, C. 2009. Primary and secondary diagnosis of multi-agent plan execution. *Autonomous Agents and Multi-Agent Systems* 18(2):267–294.

Micalizio, R. 2009. A distributed control loop for autonomous recovery in a multiagent plan. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1760–1765.

Parker, L. 1998. Alliance: an architecture for fault tolerant multirobot cooperation. *Robotics and Automation, IEEE Transactions on* 14(2):220 –240.

ProbCog. 2012. Probcog tool box. http://wwwbeetz.informatik.tu-muenchen.de/probcog-wiki/index.php/Main_Page.

Schmill, M. D.; Josyula, D.; Anderson, M. L.; Wilson, S.; Oates, T.; Perlis, D.; Wright, D.; and Fults, S. 2007. Ontologies for reasoning about failures in ai systems. In *Proceedings of the Workshop on Metareasoning in Agent Based Systems at the Sixth International Joint Conference on Autonomous Agents and Multiagent Sytems*.

Steinbauer, G., and Wotawa, F. 2009. Robust plan execution using model-based reasoning. *Advanced Robotics* 23(10):1315–1326.

Usug, U. C., and Sariel-Talay, S. 2011. Dynamic temporal planning for multirobot systems. In *Proceedings of the AAAI-11 Workshop on Automated Action Planning for Autonomous Mobile Robots (PAMR)*.

van der Krogt, R., and de Weerdt, M. 2005. Plan repair as an extension of planning. In *Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling (ICAPS-05)*, 161–170.

Wolpert, D. M., and Ghahramani, Z. 2000. Computational principles of movement neuroscience. *Nature Neuroscience* 3 Suppl:1212–1217.