Risks evaluation and Failures Diagnosis for Autonomous tasks execution in Space

Alberto Finzi, Fiora Pirri, Marco Pirrone, Massimo Romano
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza"
Roma, Italy
e-mail: {finzi,pirri,pirrone,romano}@dis.uniroma1.it

Keywords: Basic theory of actions, Perception, Failures, Diagnosis, Probability, Geons, Situation Calculus.

ABSTRACT

We present a control system for autonomous manipulators based on a theory of actions integrated with a theory of perception and failures. The theory of actions, perception and failures is defined in the Situation Calculus, a logical language that allows the representation of dynamic domains. We assume that an autonomous agent is provided with a set of possible goals. The goals are suited to the domain and the agent abilities. Despite these abilities, an autonomous agent might fail, during the execution of a task: failures can be caused by any unexpected event like loss of power, vibrations, noise, etc. These unexpected or exogenous events can cause the agent to loose an object is carrying or to slip or to hit some payload or some other structural component. While it is impossible to avoid a failure it is still possible to minimize the cumulation of failures during the execution of a task, by suitably choosing the less risky sequence of actions among all the possible sequences that allows the robot to achieve a goal. Moreover, under certain conditions it is also possible for the robot to autonomously recover from the failure. The core of the system is a high level program controlling the on-line agent behaviour: while a goal is not achieved, it selects a task from a library of possible subplans and executes it. The task, when correctly chosen, must lead to a subgoal position. At the end of each task execution, visual perception is used to monitor the coherence between the configuration of the domain and the configuration entailed as a consequence of the execution of the task. In case of a misalignment between the predicted state and the perceived one, a diagnostic procedure is activated. In our model the divergence between external information and internal representation is justified by the execution failure of the actions specified in the task. Actions are given a probability of success and since a probability can be associated also to the sentences of the logic, an expected probability of success for each goal can be computed: a goal is, in fact, represented by a sentence of the logic. This expected probability provides a measure to compute the safety of a course of actions: a course of actions is more safe than another one if it has an higher probability of success, which implies that the goal that can be achieved by such a run has a higher expected probability. Finally the diagnosis allows the robot to reset its internal status and thus to eventually recover from the error.

1. INTRODUCTION

Managing failures and safety during the execution of tasks performed by an intelligent agent is a central issue in space operations. The reason is because the cost of recover could be very high in terms of human resources. On the other hand autonomy is not only desirable but also necessary, but with autonomy the risk of failures obviously increases. It is therefore important to think about all the measures, accompanying autonomy, necessary to face failures and to ensure a flexible safety: if errors cannot be avoided, always choose those courses of actions that minimize the probability of errors, and make it possible to recover from errors. The causes of a failure can be several, and unpredictable but, depending on the task, the effects can be predictable for the agent like loosing an object it is holding, missing a grasp or sliding. The problems we want to deal with are the following:

1. Given that failures are not avoidable, ensure that a diagnosis is always possible for the agent so that it can update its current state;
2. coordinate perception and task execution, so that the agent can autonomously verify the misalignment between what it expects and what really happened, so that it can recover from the failure;
3. provide a probability of success with goal states, so that different courses of actions can be compared and a safety measure can be established;
4. embed the above requirements in a coherent dynamical system.

We present a control system for autonomous mobile manipulators based on a theory of actions and perception. The control system deals with diagnosis of possible failures, the diagnosis permits the update of the agent state to the correct situation. The system applies to autonomous manipulators built for simple missions like manipulating and moving small payloads. The core of the system is a high level program controlling the on-line agent behavior: while a goal is not achieved, it selects a task from a library of possible subplans and executes it. The task, when correctly chosen, must lead to a subgoal position. During the task, in the hypothesis that the domain is only partially observable, the agent cannot monitor its execution. In our model each action can either fail or succeed and probability of success is associated with actions, so that a task generates a stochastic process, in which also exogenous actions are considered. The computational architecture is organized into three modules: (1) A monitor, in Prolog, which is the manipulator operating system for the purpose of scheduling goals and directing calls to Golog, a high level programming language, for choosing tasks from a tasks library. (2) A local planner that expands a Golog task – a sequence of actions – into a sequence of commands to the motors, according to the existence of a free-path for the manipulator and the analytic solutions to the kinematic problems. (3) A visual module that is activated by visual sensing actions, and updates the local map of the manipulator. The computational structure operates according to the following cycle: schedule a goal, choose a task to execute, verify its executability, compute the probability of success, verify the safety of the run, even in the presence of failures, execute it, observe the state of the world modified by the execution, diagnose what happened and, if needed, update the current execution to the correct history, until the current goal is reached. The logical structure of the manipulator is formalized in the Situation Calculus [13, 19] and it is divided into an idealized, deterministic part – the basic theory of actions – and a component devoted to sensing, perception, and the non deterministic effects of actions. The basic theory of actions represents the dynamics of the world assuming all agent actions are successful. The extended theory is needed both to predict possible failures and to diagnose failures at the end of a run, under a complete failure assumption that all possible failures have been specified. Observe that also safety conditions are specified in order to suitably take into account the risks of a task execution and at which stage the verification of the current state of the execution has to take place. The formalization of perception and failures that we provide allows the system to diagnose what happened and also to project what could happen in terms of probabilities of success of each possible run, in order to accept or reject a task. The vision module is a C++ program that uses the Matrox Imaging Library. This module performs an intelligent object search in the scene with a geometric and probabilistic approach. Visual perception is used only at the end of a task and perception is, thus, used to monitor the coherence between the configuration of the domain, obtained by the agent current run and the configuration entailed as a consequence of the execution of the task. In case of a misalignment between the predicted state and the perceived one, a diagnostic procedure is activated. The diagnosis allows the agent to infer the effective run executed, selecting the most likely hypothesis that explains the perception.

2. PRELIMINARIES

We consider the Situation Calculus as the core of our logic and language, and we suitably extend it to include new sorts, new symbols for its alphabet and new axioms. See [17] for a full presentation of the core logic and language. For the extension of the language to perception and probabilities we refer the reader to [5, 6]. Here we want to recall that the Situation Calculus is a first order language in which is it possible to specify the dynamic of a system and to suitably represent the laws of change of a domain. Of particular importance are the *situations*. Situations are histories of actions: for example $goTo(pos); dock; charge(battery)$, is a sequence of actions; when this sequence has a beginning, for example in the initial situation S_0, then we have an history of what an agent has done since a given state. Another important feature of the Situation Calculus is its ability to describe the dynamic of properties in a domain, as a consequence of actions execution. A property change whenever its truth value changes, for example the light is off after it has been switched off, but before this action it was on. Properties that change are called *fluents*, which are dynamic relations, e.g. $Landing(alpha2; s)$ that specify that in the situation s the object $alpha2$ is landing, and for $s' = s$ we could also have $\neg Landing(alpha2; s')$: in other words a property holds or not depending on the situation, i.e. on the history of actions that have or not affected the property. Other important features of the Situation Calculus are the perceptibles, i.e. those observable properties of a domain that affect only the perception of the robot and not the domain, and the stochastic actions, that is, those actions with uncertain effects.

2.1 The basic ontology

An agent, e.g. a manipulator, can execute tasks and achieve goals, autonomously, only if it is endowed with some knowledge about the domain and the laws ruling the causes and effects of actions. We call all these requirements a basic theory of actions. In general, a basic theory of actions is tailored to a specific domain the one in which a robot operates. The laws of change will be different for a robotic manipulator operating in space or for a mobile robot making the tour guide in a museum. This implies that a basic theory of actions has to incorporate very specific knowledge of a domain. On the other hand this knowledge needs not to be complete, in other words, due to the dynamical structure of the logic underlying the Situation Calculus it is possible for the agent both to reason about its world even when it lacks some information and to acquire new information through perception.

Here we shall present a very simple example inspired by the exposure facilities designed for the express pallet. Our domain is that of a 7 dof robotic arm that has to manipulates payloads and to ensure their exposure for suitable experiments. We have both a simulation of the operations, that we show in Figures 1,3,4, and several implementations with small 4DOF robotic arms.

2.1.1 Descriptions

A description is needed to establish a correspondence between denotation of objects in the real world, their representation in the image and their geometric representation in a local map. For example $Land(lm_3)$ is used to denote the name of a region in the local map. In particular we shall directly use the constant ee to denote the end effector. So for example in the initial database D_{S_0} we would have sentences of the kind:

$$Payload(x) \quad ((x = pl_1 \vee \dots \vee x = pl_n) \wedge \quad (x)).$$

Where is a geometric description of the payloads in terms of our primitives categories which are the parametric geons introduced in [21].

$$Land(x) \quad = p \quad (x = lm_1 \vee \dots \vee x = lm_k).$$

$$EndEf(x) \quad x = ee.$$

Despite descriptions are not affected by world dynamics, the position of a landmark changes in the scene, even if the camera is fixed. However it is independent of the robotic actions. The basic ontology is defined specifying fluents and control actions; we also add exogenous actions that will play an important role in the extended part of the theory of actions, described later in the paper.

2.1.2 Fluents

$On(x; y; s)$: The object x is on of the object/landmark y, in situation s. To handle the projection of an object (e.g. a payload) on the map we introduce the predicate $Bottom(x; y; s)$, defined as:

$$Bottom(x; y; s) \quad On(x; y; s) \wedge Land(y) \vee \exists z.On(x; z; s) \wedge Bottom(z; y; s).$$

$Clear(x; s)$: Object x is clear, in situation s.
$Holding(x; s)$: The end-effector is holding object x, in situation s.

2.1.3 Control Actions

Figure 1: Simulation of the exposure facility: lifting a payload

$moveTo(x, y)$: While holding object x, the robot moves it to a reference object/landmark y.
$goTo(x)$: The end-effector moves to reference object/landmark x.
$grasp(x)$: Grasp the object x.

2.1.4 Exogenous Actions

$slip$: The robot slips, which means that the end-effector is no longer in the expected position.

$fall(x)$: The object x falls and is no longer in its original position.
$shift(x, y)$: Object x shifts from its current position to position y.
These are all the possible control and exogenous actions, and their effects are those explicitly mentioned. Observe that this last clause is a *completeness assumption for failure*: any change in the domain is due to the effect of either a control action or an exogenous actions; nothing else affects the world.

2.1.5 Successor State Axioms

An example of a successor state axiom for the basic ontology is the following:

$$\begin{array}{l} On(x, y, do(a, s)) \equiv a = moveTo(x, y) \lor a = shift(x, y) \lor \\ \quad On(x, y, s) \land \\ \quad (a = grasp(x) \land a = fall(x) \land a = slip \land \\ \quad \forall z.(z = y \rightarrow (a = shift(x, z) \land a = moveTo(x, z)))). \end{array}$$

3. BEYOND THE BASIC ONTOLOGY

3.1 A model for perception

For a detailed presentation of perception in the Situation Calculus we refer the reader to [16]. In the framework of the robotic manipulators, we manage the perception of the scene by a single sensing action taking as arguments the primitive perceptibles, which for the *express pallet world* are $isLand$, $isPayload$; $isEndEf$ as follows:

$$observeScene(isP_1(\vec{x}), \ldots, isP_n(\vec{x}), pr_1, \ldots, pr_n)$$

Here each isP_i is a primitive perceptible and the pr_i's are outcomes of visual perception: $pr = 1$ means that the perceived property, denoted by the perceptible, holds in the scene, and 0 that it does not. In this framework the scene is a fluent taking as argument a primitive perceptible and a situation. The following successor state axioms account for the primitive perceptible explicit definition and description with respect to the scene:

$$\begin{array}{l} Scene(isPayload(x), do(a, s)) \equiv a = \\ observeScene(isPayload(pl_1), \ldots, pr_1, \ldots, pr_n) \land \\ (x = pl_1 \land pr_1 = 1 \lor \ldots \lor x = p_k \land pr_m = 1) \land \\ Descr(isPayload(x)) \lor Scene(isPayload(x), s) \land \\ \forall p_1 {:}{:} p_n\, pr_1 {:}{:} pr_n . a = observeScene(p_1, \ldots, p_n, pr_1, \ldots, pr_n) \end{array}$$

Here $Descr(isPayload(x))$ is a suitable description of a payload, e.g. containing information about the dimension, shape, weight etc. In this framework, given that a single sensing action over the primitive perceptibles is defined, the percepts are used for constructing the relations among primitive perceptibles and for recording the perceptible and their outcomes at each situation:

$$\begin{array}{l} Percept(isOn(x, y), pr, s) \equiv x = y \land \\ \quad (pr = 1 \land \\ \quad\quad Scene(isPayload(x), s) \land (Scene(isPayload(y), s) \lor \\ \quad Scene(isLand(y), s)) \land Descr(isOn(x, y))) \lor \\ \quad (pr = 0 \land \\ \quad \neg(Scene(isPayload(x), s) \land (Scene(isPayload(y), s) \lor \\ \quad Scene(isLand(y), s)) \land Descr(isOn(x, y)))). \end{array}$$

Here $Descr(isOn(x, y))$ is a description of the relation $isOn$. After an observation of the scene there might be a discrepancy between what is perceived and what is entailed by the database: the discrepancy will not cause a logical inconsistency because it is recorded as, e.g. $On(a, b, s)$ and $Percept(isOn(a, b), 0, s)$.

These discrepancies are detected by a defined predicate $Mistaken(p, s)$, for each perceptible p, which will be used to diagnose what the real effects of the actions performed by the manipulator are. For an analysis of the different levels of perception, from direct perception to selective sense perception, to meaningful perception and how mistakes are treated, we refer the reader to [16].

3.2 A model for failures

In the basic ontology we have defined an idealized representation of the world in which the effects of actions are exactly those intended, i.e. actions, even exogenous actions, are deterministic. However, due to various circumstances (e.g. a sudden change in the power supply), the robot might fail in its intended execution of an action. To address this problem, we have extended the specification of an action theory with stochastic actions and events, which we briefly present here.
A *stochastic action* is a pair $\langle v(a), a\rangle$ with $v(a)$ taking values in the outcome space $outcome = \{0, 1\}$, where 0 means that a failed and 1 that it succeeded. Given a sequence of actions, this sequence expands to a tree of events due to all the possible outcomes of each action. See Figure 2. To model the tree and the probability distribution we introduce the following notion of event.
Event. An event e is either a sequence event W of sort $seqEvent$ or a conditional event u of sort $condEvent$, according to the following definitions:
sequence event:

1. E_0 : the empty event.
2. $(\langle v(a), a\rangle \circ W)$: the event obtained as a result of executing a with value $v(a) \in \{0, 1\}$, after the sequence event W.

conditional event

3. $(\langle v(a), a\rangle \mid W)$: the event obtained as a result of executing a with value $v(a) \in \{0, 1\}$, conditioned on the sequence event W.

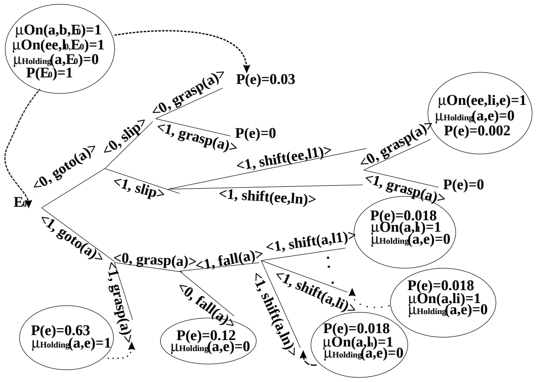

Figure 2: The tree $\mathtt{tree}(\mathtt{do}(\mathtt{goTo}(a); \mathtt{do}(\mathtt{grasp}(a); S_0)))$

Therefore an event is either a sequence of outcomes of stochastic actions or the outcome of a stochastic action conditioned on a non conditional event. Accordingly, we extend the notion of fluents to *event fluents*, that is fluents that have as arguments events.
Probability is a functional event fluent **P** which takes as arguments events and returns their probability: **P** maps events into $[0; 1]$.

Figure 3: Dealing with an experiment

3.2.1 *Axioms for events and probability* P

The axioms for event have been given in , they are needed to ensure that the initial situation S_0 coincide with the initial event E_0 and that events branch as a tree, rooted in E_0, see Figure 2. Here we recall the axioms and basic properties for probability **P**, more can be found in [6].

1. $\mathbf{P}(e) \quad 0$, for any event e.

2. $\mathbf{P}(E_0) = 1$.

3. $\mathbf{P}(sta \quad e) = \mathbf{P}(sta \mid e) \quad \mathbf{P}(e)$

4. $\mathbf{P}(\langle 0; a_i\rangle \mid e) = 1 \quad \mathbf{P}(\langle 1; a_i\rangle \mid e)$.

Let $\quad \in \{0; 1\}$. We define the following functions:

$$\mathtt{sit}(E_0) = S_0$$
$$\mathtt{sit}(\langle \; ; a_i \rangle \quad e) = \mathtt{do}(a; \mathtt{sit}(e))$$
$$\mathtt{outcome}(\langle \; ; a_i \rangle \quad e; \langle \; ; a_i\rangle) = \quad :$$

The following sets can be defined using $\mathtt{sit}$, with $\mathtt{ExAct}$ a predicate that sorts out the exogenous actions:

$$\mathtt{tree}(s) = \{\langle \; ; a_i \rangle \quad W \mid \mathtt{ExAct}(a) \wedge W \in \mathtt{tree}(s) \vee \mathtt{sit}(\langle \; ; a_i\rangle \quad W) \vee s\}$$
$$\mathtt{cut}(s) = \{W \mid W \in \mathtt{tree}(s) \wedge \forall W' : W' \in \mathtt{cut} \rightarrow W \not\subset W'\}$$
$$\mathtt{leaves}(s) = \{W \mid W \in \mathtt{tree}(s) \wedge \neg\exists W' : W' \in \mathtt{tree}(s) \wedge W \quad W'\}$$

3.3 Preconditions of stochastic actions and conditional probabilities

Given a sequence of stochastic actions $W = \langle v(\;1); \;1\rangle \; ::: \; \langle v(\;n); \;n\rangle \quad E_0$, a stochastic action $sta = \langle v(a); a\rangle$ could be executed with success in W if suitable conditions are satisfied. The following schema captures this fact:

$$\mathtt{Could}(\langle 1; a\rangle; W) \qquad _{\langle 1; a\rangle}(1; a; W)$$

where specifies the conditions for executing the action a with outcome 1. The above axiom states which preconditions have to be satisfied, at a given sequence of stochastic actions W, for the action to be executable with success. For example, if the robot intended to go to a given position and it slipped, it follows that the measure of the fluent $\mathtt{On}(ee; lm_3)$ cannot be 1 and the action $\mathtt{grasp}(a)$, if a is on landmark 3, cannot be executed with outcome 1. The above preconditions are also needed to define the probability of success of a stochastic action. We show, in the sequel how we define the axioms for the preconditions of stochastic actions, in the express pallet world, in which initial information might have been acquired through perception. Consider the control and exogenous actions given in paragraph *2.1.3* and *2.1.4*.

$$\mathtt{grasp}(x); \mathtt{goTo}(x); \mathtt{moveTo}(x; y); \mathtt{fall}(x); \mathtt{slip}; \mathtt{shift}(x; y)$$

The preconditions for $\mathtt{grasp}$ and for the exogenous action $\mathtt{fall}$ are:

$$\mathtt{Could}(\langle 1; \mathtt{grasp}(x)\rangle; e) \quad \mathtt{Payload}(x) \wedge \mathtt{Clear}(x; e) = 1 \wedge$$
$$\mathtt{On}(ee; x; e) = 1 \wedge \forall z : \mathtt{Payload}(z) \rightarrow \mathtt{Holding}(z; e) = 0:$$
$$\mathtt{Could}(\langle 1; \mathtt{fall}(x)\rangle; sta \quad e) \quad sta = \langle 0; \mathtt{grasp}(x)\rangle \vee$$
$$\exists y : sta = \langle 0; \mathtt{moveTo}(x; y)\rangle:$$

Observe that the preconditions for the exogenous actions to be successfully executed depend also on the failure of previous control actions. So, for example, the end-effector can slip just in case either the action $\mathtt{goTo}$ or the action $\mathtt{moveTo}$ just failed. Therefore the conditional probability of the exogenous action are conditioned on *random effect* of stochastic actions. The following is an example of the conditional probabilities of the exogenous actions $\mathtt{fall}(x)$, conditioned on the failure of a stochastic action:

$$\mathbf{P}(\langle 1; \mathtt{fall}(x)\rangle \mid sta \quad e) = p \quad \mathtt{Could}(\langle 1; \mathtt{fall}(x)\rangle; sta \quad e) \wedge$$
$$(sta = \langle 0; \mathtt{grasp}(x)\rangle \wedge p = 0{:}6) \vee$$
$$(sta = \langle 0; \mathtt{moveTo}(x; y)\rangle \wedge p = 0{:}4) \vee$$
$$p = 0 \wedge \neg\mathtt{Could}(\langle 1; \mathtt{fall}(x)\rangle; sta \quad e):$$

Observe that by the axioms for **P** we have for each of the above stochastic actions also a conditional probability for failure:

$$\mathbf{P}(\langle 0; a\rangle \mid e) = 1 \quad \mathbf{P}(\langle 1; a\rangle \mid e)$$

3.4 Updating the knowledge base and diagnosing failures

At the end of the execution of a task the action $\mathtt{observeScene}$ is performed and therefore for each property in the world there will be a perceptible with outcome 0 or 1. For example, if a is on b, then the robot will infer $\mathtt{Percept}(\mathtt{isOn}(b; a); 1; s)$,

Figure 4: Concluding the task.

where s is the current situation after the observation of the scene. Perception, might disagree with what the agent's database entails. For example if the agent did the action $moveTo(b, lm_3)$ but it failed to pick up b then the agent database would entail $On(b, lm_3)$, and so contradict the result of perception. However, on the stochastic tree, there is an event telling the true story, namely, that the agent failed to grasp b. To find this event we introduce the notion of likelihood, that will allow us to select the most likely event that happened and that can explain the current perception.

3.4.1 Likelihood

Let σ be a sequence of actions. Let $W = sta_1 \ldots sta_k \quad E_0 \in leaf(tree(\sigma))$ be a final sequence of stochastic actions.

$$
\begin{array}{l}
likelihood(W) = 1 \quad \stackrel{def}{\equiv} \mathbf{P}(W) = 0 \wedge \\
\quad \forall x\, y{:}Payload(x) \wedge (Payload(y) \vee Land(y)) \rightarrow \\
\quad Percept(isOn(x, y), 1, \sigma) \rightarrow (On(x, y, W) \wedge \\
\quad \forall x{:}((Percept(isHolding(x), 1, \sigma) \rightarrow Holding(x, W)) \wedge \\
(Percept(isHolding(x), 0, \sigma) \rightarrow \neg Holding(x, W)) \wedge \\
\quad Percept(isClear(x), 1, \sigma) \rightarrow Clear(x, W).
\end{array}
$$

Observe that the fluents above take as argument a sequence-event, as they are belief (see [6] for more details). We define, now, a preference relation $<$ between stochastic sequences as follows:

$$
\begin{array}{l}
W < W' \quad \stackrel{def}{\equiv} \\
\quad likelihood(W) < likelihood(W') \wedge \mathbf{P}(W) < \mathbf{P}(W').
\end{array}
$$

Note that $<$ induces a partial order. Consider the set of $<$-maximal elements. We can then define a heuristic to choose the most likely stochastic situation. For this simple domain we have chosen to prefer a sequence W to a sequence W' if the number of occurrences of exogenous actions in W is less than in W' and the sequence of 0's in the outcomes of W are less that in W'. These preferences are reasonable and, in general, enough to identify a unique preferred sequence W of stochastic actions in the set of $<$-maximal elements.

Once the most likely sequence W is computed, according to a preference criterion , then the sequence W is a correct sequence of stochastic control and exogenous actions that could have been "effectively" executed – under a complete failure hypothesis, i.e. all the possible failures and exogenous actions have been represented.

Let σ be a sequence of actions, let $W_{max} \in tree(\sigma)$ be the sequence of stochastic actions satisfying the maximum likelihood hypothesis according to the preference criterion and the heuristic. Let $\sigma' = sit(W_{max})$. There exists a sequence σ^*, which can be obtained from σ', such that σ^* *explains* the perception in σ:

THEOREM 1. *Let W be a sequence of stochastic actions satisfying the maximal likelihood established by criteria in . Then there exists a sequence σ^* and a function $trans$ such that:*

$$
\begin{array}{l}
D \models sit(W) = \sigma \wedge \sigma^* = trans(W) \wedge \\
\quad \forall x\, Percept(isP(x), 1, \sigma) \rightarrow P(x, \sigma^*)
\end{array}
$$

Here $trans$ is a function eliminating from the event W all the stochastic actions having outcome 0.

3.5 Expected Probability and Safety

In [5] we have proposed a formal methodology to compute the probability of sentences in the initial situation D_{S_0}, under the constraint that the domain of world entities has a fixed cardinality. Now, given a sequence of actions σ, where σ could be a sequence like $moveTo(payload1); insertGripper, lift(payload1); move\text{-}On(pl1, pl2)$, and a goal G that requires the sequence σ to be achieved, for example G could be $On(pl1, pl2, \sigma)$, we want to determine the probability of $G(\sigma)$, given that each action can succeed with a given probability. To this end we need to combine the logical probability on sentences and the event probability defined on stochastic actions to get a third probability that accounts for both incomplete information and possible failures of actions during execution. To capture these ideas we introduce the notion of *Expected Probability*. Let $E_{set}(\sigma', \sigma) = \{e_i : e_i \in leaves(\sigma), trans(e_i) = \sigma'\}$. Let $T(\sigma) = \{\sigma' \mid trans(e_i) = \sigma', e_i \in leaves(\sigma)\}$, and let R be the regression operator, see [?].

DEFINITION 1 (EXPECTED PROBABILITY IN S_0). *Let φ and be sentences uniform in σ. The expected probability, in S_0, of φ, given evidence is:*

$$
EP(\varphi(\sigma) \mid (\sigma)) = \sum_{\sigma' \in T(\sigma)} (Prob(R(\varphi(\sigma')) \mid R((\sigma))) \sum_{e_i \in E_{set}(\sigma', \sigma)} \mathbf{P}(e_i))
$$

The above defined expected probability allows us to compare two sequences of actions and decide which one is safer:

DEFINITION 2 (SAFETY). *Let $\varphi(\sigma_1)$ and $\varphi(\sigma_2)$ be two sentence uniform, respectively, in σ_1 and σ_2. Let $\preceq_{safe}$ be a ordering relation:*

$$
\varphi(\sigma_1) \preceq_{safe} \varphi(\sigma_2) \text{ iff } EP(\varphi(\sigma_1)) \leq EP(\varphi(\sigma_2))
$$

The notion of *Safety* is meaningful because it allows to compare two runs and decide which one is more reliable, just from the point of view of what the robot knows in the initial situation and on the probability of success of each action mentioned in the run. Given a set of goals that the robot has to achieve, it is possible to assign a reinforce to all those states in which the goal is satisfied, according to the probability of success and the safety of the sequence leading to such a state. The reinforce will make it possible to define suitable policies, i.e. courses of actions for each goal, that the robot should take, in order to achieve all the goals that have been scheduled.

Observe, however, that a robot cannot backtrack from a chosen Goal situation, therefore in maximizing the reward from all the goal states it has to take into consideration the fact that the next goal can be reachable from the current situation. We are still developing this approach, and in our implementation the choice of a task is still non deterministic.

4. IMPLEMENTATION

4.1 Perception

The role of perception is to correctly instantiate the sensing action `observeScene` and to update the local map.

The perceptual system is composed of a cognitive part, that manages reasoning about perception and perceptibles, a recognition part and an analytic part. The core of the system is formed by the concept of parametric Geons [21]. Parametric geons are seven volumetric shapes, obtained by specifying the shape parameters in a superquadrics equation and applying tapering and bending deformations, see Figure 5

Figure 5: Parametric Geons

The cognitive part is formalized in the theory of actions via *Descriptions* and axioms for perceptibles: for each category of objects or property or relation which is expected to be observed in the domain, a suitable sentence states the conditions under which it can be sensed and acquired at the cognitive level, in order to reason about it. The recognition part is formalized via a Bayesian network in which 3D geometric primitives, i.e. parametric geons [21], are inferred using evidence available from 2D image features and arcs of lines. The Bayesian Network is also used, together with the cognitive level, to infer a plausible reconstruction of an object in the scene. At the lower, analytical level, a neural network elaborates the arc segments obtained at the end of the process that delivers the edges and contours of the objects in the scene, and releases the suitable features needed by the Bayesian Network. In Figure 6 some of the nodes of the Bayesian Network are displayed, without the conditional tables and the connections: at the lower levels are the categorizations obtained by suitably elaborating the neural network output. Bayesian Networks have already been used in vision, by Jensen et al [4] for context based modeling and interpretation. In particular Fairwood and Barreau [11] proposed the use of a Bayesian network to infer 3D geometric primitives (geons) using evidence available from 2D image feature detectors. Moreover Sarkar and Boyer [20] developed the Perceptual Inference Network (PIN), an extension of the Bayesian network, that was used to organize knowledge about low-level features, such as edges and corners, to reason about higher level features.

We model the scene with a hierarchical geometric model whose components are the connections among the entities in the scene and their properties. Entities are instantiated with features, e.g. color, orientation w.r.t. the reference frame, etc. The known entities are represented in a *normalized form*.

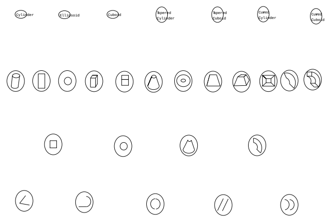

Figure 6: The entities of the Bayes Network.

The vision module is a C++ program realized in the ALCOR laboratory using the Matrox Imaging Library. This module performs an intelligent object search in the scene with a geometric and probabilistic approach. It executes the following tasks: image acquisition, noise reduction, scene segmentation, edge detection, vertex recognition and 3D reconstruction.
We have been using a couple of ultra small CCD color video cameras XC-999, by RWII, with high resolution and definition.

4.2 High Level Control

The high level control is managed by a Golog programs. Golog [12] is a situation-calculus based logic programming language suitable for the high level agent control. This language employs the primitive actions specified in the knowledge base, and provides the following standard, and not so standard, Algol-like control structures:

1. Sequence : $\quad :$
2. Test action : $\quad \phi?$
3. Nondeterministic choice : $\quad |$
4. Nondeterministic choice of arguments : $\quad (\pi x)$
5. While loop :
6. Loops and Procedures, including recursion.

Golog's semantics is specified by means of a *macro* $Do(\ ; s; s')$, where is a program, and s and s' are situation terms. The macro abbreviates a formula ϕ of the Situation Calculus, which interprets the program as a sequence of actions that should be executed to transit from s to s'. Therefore, each program execution corresponds to a Situation Calculus formula. This correspondence is defined inductively as follows:

primtive action : $Do(\ ; s; s') \overset{def}{=} poss(a; s) \wedge s = do(a; s)$
test : $Do(\phi?; s; s') \overset{def}{=} \phi[s] \wedge s = s'$
sequence : $Do(\ :\ ; s; s') \overset{def}{=} \exists s\ Do(\ ; s; s\ ) \wedge Do(\ ; s\ ; s')$
nondet: choice : $Do(a|b; s; s') \overset{def}{=} Do(a; s; s') \vee Do(b; s; s')$
nondet: choice of arg: : $Do((\pi x)\ ; s; s') \overset{def}{=} \exists x Do(\ ; s; s')$

Both procedure calls and blocks with local procedure declarations can be defined; for details see [12].

The high level primitive actions and the non deterministic constructs allow for the definition of plan schemas suitable for the description of agent's behaviors. To adapt a Golog execution to the space of events, we extend Golog with the following new construct:

$$
\begin{aligned}
&Do(call(X); s; s') = \exists e; e': sit(e) = s \wedge \\
&\quad \forall sta\ e''(e''\ \ e \wedge e'' = E_0 \rightarrow outcome(sta\ \ e''; sta) = 1) \wedge \\
&\quad Do(X; s; s') \wedge sit(e') = s'.
\end{aligned}
$$

where $sit(e)$ and $outcome(sta\ \ e; sta)$ are the functions defined in Section **??**. To connect the high level executability of a program to the low level executability of the primitive operations of the manipulator we introduce the following macro.

$$DoG(prog, s, s') \stackrel{def}{=} Do(\ , s, s') \wedge exec(s, s')$$
where
$$exec(s, s') \quad \forall s'' a. s \sqsubseteq do(a, s'') \sqsubseteq s' \rightarrow PossG(a, s'')$$
and
$$PossG(a, s) \stackrel{def}{=} Poss(a, s) \wedge findPath(a, s)$$

Observe that $exec(s, s')$ states that the trace of the program is *low level executable*, $PossG$ is a guarded $Poss$, and $findPath$ represents a guard for the action preconditions verifying the low level executability of the action a.
The Golog variant that we have just presented allows for the implementation of the high level controller. The monitor, in fact, is based on the following Golog procedures.

proc chooseTask;
  goal | [(¬goal)?; π(X)(task(X); observeScene)]
endProc
proc verify(S); call(verify(S)) **endProc**

The procedure **chooseTask** consists of a non deterministic choice of a task task(X) followed by an observation of the scene. A task represents a specific sequence of actions that the agent intends to execute. The task is chosen just in case it is executable at all the layers of the agent architecture: a task fails if its global or local expansion fails. After the blind execution of the task the observeScene action gathers the perceptual information about the current *external* state of the world. The procedure verify(s) is then called to get, among all the possible ones, the most plausible execution that can explain the observed state of the world. This procedure consists of a procedure call in the space of the events employing the construct call. At the end of the procedure a new final situation is reached; this situation explains what has been perceived. The following tasks library accounts for all the possible behaviors of the agent at its Express Pallet workspace.

proc task(1);
  π(X)(π(l)((clear(X) ∧ ¬onPallet(x, l)∧
  ¬goodTower(X))?) ; goTo(X) ; grasp(X) ;
  π(l1)(moveTo(x, l1); land(l1)?) ; goTo(l0))
endProc

proc task(2);
  π(X)(π(y)((clear(X) ∧ clear(y)∧
  onPallet(X) ∧ goodTower(y) ∧ sameColor(x, y))? ;
  goTo(X) ; grasp(X) ; moveTo(x, y) ; goTo(l0)))
endProc

proc task(rec);
  π(X)((object(X) ∧ badPosition(X))?;
  goTo(X) ; grasp(X) ; π(y)((land(y)∧
  ¬badPosition(y))? ; moveTo(x, y)))
endProc

onPallet(x, s) is an abbreviation for: ∃l(land(l) ∧ on(x, l)). task(1) is used to unstack a bad tower (i.e. a tower not satisfying the expected goal conditions). Therefore, it requires that each object that can be grasped and belongs to a bad tower must be moved to a landmark. task(2) is used to construct good towers, here goodTower(X) is a heuristic definition describing towers satisfying possible goal conditions: each object that can be moved to a good tower is moved. The task task(rec) is used to remove an object from a bad position, also badPosition(X) is a heuristic definition corresponding to any position obstructing the path. At the end of each of the above tasks the robot, if had been successful during the execution, should have reached a subgoal. If not then a plausible explanation of what had happened has to be determined. To search for the most plausible explanation of the observation performed by the action observeScene the following procedures are executed in the events space.

proc verify(s); generateEvent(s); likelihood(s)?
endProc
proc generateEvent(s);
  (s = s0)? | π(s')(π(a)((s = do(a, s'))?;
  generateEvent(s'); stocasticActions(a)))
endProc
proc stocasticAction(a); h1; aijh0; ai; compAction
endProc
proc compAction;
  π(X)(exogenousAction(X)? : h1; aijh0; ai)
endProc

The procedure verify amounts to a nondeterministic search, in the space of the events, for that event satisfying a maximum likelihood value, which represents the most plausible explanation. Observe that, as noted above, a set of preference measures is used until the likelihood converges to a single event state. A Golog procedure must be translated into suitable low level primitive operations for the functional architecture of the manipulator, before the execution; therefore it is necessary to define a monitor, the robot operating system, guiding both compilation and execution of Golog programs. This monitor program cycles between first compiling and executing the procedure **chooseTask**, and compiling and executing the procedure **verify(s)** until the goal is achieved. Golog programs are translated into the low level primitive operations, evaluating the formula $DoG(\ , s, s')$ checking low level executability. A monitor cycle works in the following way: if the start situation is a goal situation then the monitor stops and it schedules another goal. Otherwise, a **chooseTask** is compiled and translated into the local level for evaluation and execution. After the execution, the *verify* procedure is performed to get the situation that explains the perceived state. The monitor is implemented as a Prolog program.

5. DISCUSSION

In this paper we have presented a control system for autonomous mobile manipulators managing failures of actions and perception.
In our approach, actions are considered as non-deterministic and stochastic; for each action, the probability of success and failure is given. To manage failure, we have adopted a probabilistic approach that is similar to that proposed by Bacchus, Halpern and Levesque in [1], where a model s presented for reasoning about an agent's probabilistic degree of belief. In that framework, noisy sensors and effectors are modeled using stochastic actions that affect the beliefs of the agent. In contrast to our model, where the sensors are sources of "truth" used to evaluate the beliefs of the agent, in their approach a direct comparison with the "real world" is not considered because all the reasoning resides in the "mind" of the agent.
The problem of combining actions and probabilities has been considering in the important work of Poole [18] that presents a way to combine decision theory, situation calculus and conditional plans. Unlike [1]'s approach, Poole's differs from ours in many respects. The main difference is that in our frame-

work, a probability distribution is assigned to the actions (that may succeed or fail) while in Poole's approach the probability is associated with the state. Reiter [19] also considers the representation of stochastic actions in an action theory, by decomposing a non-deterministic action into deterministic components selected probabilistically by nature. He does not give an explicit formalization for probabilities. Another interesting probabilistic model for dynamic domains can be found in [14, 15] in the context of reactive planning [2]. This probabilistic model differs from ours in several respects. The ontology underlying the dynamics is temporal. Furthermore the probability of events is context dependent, while we consider that probabilities depend on histories of stochastic actions.
A probabilistic representation is also employed by RHINO [3], but at the lower level. This representation is used for the sensor model, which can be inaccurate and incomplete, while we assume that visual perception is reliable and complete, for the domain at hand. The crucial difference is in the fact that we are dealing with small mobile manipulators, not mobots. Therefore, our environment might be complex – which could even be more informative for localization – but is quite static and limited: this also allows the manipulator to have an external perspective from which an objective and total view of the environment can be got. At the task level, RHINO employs high level Golog programs executed by GOLEX [9]. Our solution is different; we have a high level execution monitor that manages compilation and execution of Golog programs and the execution is monitored after a sequence of actions, while GOLEX monitors after each action. In our framework failures and perception are represented and treated at the logical level, while in RHINO they are treated dynamically by the monitor at the lower level. In this respect, the novelty of our approach is in our uniform formal framework, allowing us to infer explanations for failures; this explanation is the recovered situation.
A high level programming approach is a central issue also for Funge [7] that addresses the domain of computer games introducing cognitive modeling for autonomous characters animation. Here failures are managed at the high level using sensing actions (implemented using interval-valued-epistemic-fluents [8]). Sensing action outcomes are used to test action effects and for replanning or recovering. In our system, perceptual information is used simply to explain failures, after which the monitor restarts from a plausible state of the world obtained by the explained situation. Failure recovery is an important topic in the ROUGE architecture [10]. In this framework, monitoring is performed after each action execution and perception allows the system to adapt to its environment, making relevant planning decisions. Also here failures are treated dynamically, detecting and compensating for them at run time. A model for failures is not employed; failure detection directly induces an update in the domain knowledge, and that initiates the replanning activity.

Acknowledgment

The paper describes research done at the ALCOR laboratory. Support for the laboratory's research is provided in part by the ASI (Italian Space Agency) project PEGASO (PErcept Golog for Autonomous agents in Space station Operations).

6. REFERENCES

[1] F. Bacchus, J. Halpern, and H. Levesque. Reasoning about noisy sensors in the situation calculus. *Artificial Intelligence*, 111:171–208, 1999.

[2] M. Beetz and D. McDermott. Declarative goals in reactive plans. In J. Hendler, editor, *Declarative goals in reactive plans*, pages 3–12, 1992.

[3] W. Burgard, A. Cremers, D. Fox, D. Hahnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Journal of Artificial Intelligence*, 114(1–2), 1998.

[4] R. Fairwood and G. Barreau. A belief network for the recognition of 3-d geometric primitives. In *Sixth International conference on Image Analysis and Processing*, pages 363 – 370, 1991.

[5] A. Finzi and F. Pirri. Combining probabilities, failures and safety in robot control. In *In Proceedings of IJCAI-01*, page to appear, 2001.

[6] A. Finzi, F. Pirri, M. Pirrone, and M. Romano. Autonomous mobile manipulators managing perception and failures. In *In Proceedings of Autonomous Agents 2001*, page to appear, 2001.

[7] J. Funge. Cognitive modeling for games and animation. *Comunications of the ACM*, 43(7).

[8] J. Funge. Representing knowledge within the situation calculus using intervalvalued epistemic fluents. *Journal of Reliable Computing*.

[9] B. W. Hahnel, D. and G. Lakemayer. *GOLEX-Bridging the Gap between Logic (GOLOG) and Real Robot*. 1998.

[10] K. Z. Haigh and M. M. Veloso. High-level planning and low-level execution: Towards a complete robotic agent. In W. L. Johnson, editor, *Proceedings of First International Conference on Autonomous Agents*, pages 363–370. ACM Press, New York, NY, 1997.

[11] F. V. Jensen, H. I. Christensen, and J. Nielsen. Bayesian methods for interpretation and control in multi-agent vision systems. In *Proceedings from Applications on AI X: Machine Vision and Robotics*, pages 536 – 548, 1992.

[12] H. Levesque, R. Reiter, Y. Lesperance, F. Lin, and R. Scherl. Golog: A logic programming language for dynamic domains. *Journal of Logic Programming*, 31:59–84, 1997.

[13] J. McCarthy. *Situations, actions and causal laws*. MIT Press, Cambridge, MA, 1969.

[14] D. McDermott. *Probabilistic Projection In Planning*. Dordrecht: Kluwer Academic.

[15] D. McDermott. Using regression-match graphs to control search in planning. *Artificial Intelligence Journal*, 109(1–2):111–159.

[16] F. Pirri and A. Finzi. An approach to perception in theory of actions: Part I. *ETAI*, 4, 1999.

[17] F. Pirri and R. Reiter. Some contributions to the metatheory of the situation calculus. *ACM*, 46(3):261–325, 1999.

[18] D. Poole. Decision theory, the situation calculus, and conditional plans. *ETAI*, 3(8), 1998.

[19] R. Reiter. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. MIT press. To appear, www.utoronto.ca/ cogrobo, 1998.

[20] S. Sarkar and K. L. Boyer. Integration, inference and management of spatial information using bayesian networks: Perceptual organization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(3):256–274, 1993.

[21] K. Wu and M. Levine. 3D object representation using parametric geons. Technical Report CIM-93-13, 1993.