Autonomous Mobile Manipulators Managing Perception and Failures

Alberto Finzi, Fiora Pirri, Marco Pirrone, Massimo Romano, Milko Vaccaro
Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, 00198, Roma, Italy
{ finzi,pirri,pirrone,romano,vaccaro } @dis.uniroma1.it

ABSTRACT

We present a control system for autonomous mobile manipulators based on a theory of actions integrated with a theory of perception and failures. The system applies to autonomous manipulators built for simple missions like building towers of blocks, collecting balls of paper and putting them into a bin, or clearing obstacles from a path. For these application domains we have built three autonomous agents: ArmHand0, ArmHandONE and A.R.R.I.G.O. The core of the system is a high level program controlling the online 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.

1. INTRODUCTION

In this work we present a control system that has been implemented on three manipulators, differing from one another by the shapes of their grippers and whether or not is it possible to control the wheels, which are features tailored for their missions: the blocks world, the ball of paper collector and the obstacle-remover Because of the system's complexity and limited space we can only sketch its formalization and implementation. The basic ideas are the following. We have a computational architecture, whose modules use Golog [11], Prolog, C++ and MatLab. The logical and algebraic architecture is based on the Situation Calculus [12, 19] and integrates a basic theory of actions with perception and action failures. 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 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 the single action $observeScene$ 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, 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 [12, 19] and it is divided into an idealized, deterministic part – the basic theory of actions – and a component devoted to sensing, formalizing 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 represents what really happened, under the *complete failure assumption* that all possible failures have been specified.

Suppose that a sequence of actions, like moving the arm to block a and grasping it, is executable (meaning that each action precondition is satisfied). During execution, it might happen that the gripper slips and the block falls to the table. Then at the end of the sequence, the block will be on the table, while the manipulator expects to be holding it. The formalization of perception and failures that we provide allows the system to diagnose what happened, e.g. that the gripper slipped and the block fell over, and to update the action history based on its diagnosis.

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. To better explain our extensions, we give a few details about the Situation Calculus. The logical structure is a many sorted first order logic with equality defined by a signature $SC = \langle Sort, Fun, Pred, Var\rangle$, a language $\mathcal{L}_{SC}$ (the language of Situation Calculus) and an interpretation $\mathcal{I} = (D, \_^{Fun}, \_^{Pred})$. $Sort$ is a set, each element of which is a sort. The core $Sort$ consists of three sorts: *action, situation* and $object$. We have extended the core with four new sorts: *perceptible, outcome, stochasticAction, event*. The sort event consists of two subsorts, $event = condEvent \cup seqEvent$ and the sort object is a catch-all that can include any possible additional sorts needed in an application, like the reals or sets. Fun is a family of sets including, for example, functions of sort object e.g $\{+, -, /, length, ..\}$, functions of sort action e.g. $\{grasp, goTo, fall, slide, ..\}$, functions of sort perceptibles e.g. $\{isOn, isGreen, ..\}$, and so forth: each element in a set of Fun is a function symbol whose sort is in $Sort$ and having as arguments terms which, in turn, belong to $Sort$. $Pred$ is a family of sets including, for example, relations

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
AGENTS'01, May 28-June 1, 2001, Montréal, Quebec, Canada.
Copyright 2001 ACM 1-58113-326-X/01/0005 ...$5.00.

taking as arguments terms of sort object and terms of sort situation, which are called relational fluents (i.e. dynamic relations, because their truth values depend on their situation argument) like $\{Clear, On, OnTable, ..\}$; the relation $\{Percept\}$, which takes as argument perceptibles and terms of sort situation; non-dynamic relations like $\{Table, Block, Set, Number, ..\}$. Each element in a set of $Pred$ is a predicate symbol having as arguments terms, which belong to $Sort$. Var is a family of sets. Finally D is a sorted domain. Observe that D could be infinite or uncountable even if some of its subset is finite. For example we would be interested to impose a finite domain for the denotation of the objects in the scene, while we would require probabilities to range over the reals. We shall use the following alphabet, with subscripts and superscripts, to denote elements of Var: a, for variables of sort action; s, for variables of sort situation; sta, for variables of sort stochastic action; e, for variables of sort event; w for variables of the sub-sort $seqEvent$ and u for variables of the sub-sort $condEvent$; p, for variables of sort perceptible; pr, for variables of sort outcome and $x, y, z...$, for variables of sort object. Each of the above sets $Sort, Fun, Pred$ and Var are pairwise disjoint.
The family of terms over SC and the set of formulae $\mathcal{L}_{SC}$ over SC are inductively defined as usual. The following constants, functions and relations are of particular relevance:

The core alphabet:

1. *action*. $a : (action \cup object)^n \mapsto action$, e.g. $grasp(block1)$.
2. *situation*. $do : action \times situation \mapsto situation$, and the special constant S_0.
   E.g. $do(grasp(block1, do(goTo(table, block2), S_0)))$ is a situation.
3. $\sqsubset$: $Sit \times Sit \mapsto Sit$, e.g. $S_0 \sqsubset do(grasp(block1, S_0))$.
4. *Poss*. $action \times situation$, e.g. $Poss(grasp(block1), S_0)$.
5. *functional fluent*. $f : (action \cup object)^n \times situation \mapsto action \cup object$.
   E.g. $distance(block1, block2, S_0) = 3.4$.
6. *relational fluent*. $(action \cup object)^n \times situation$, e.g. $Clear(block1, S_0)$.

The extended alphabet:

7. *sense action*. $sense : (perceptible \cup outcome) \mapsto action$, e.g. $sense(isClear(block1), 1)$.
8. *perceptible*. $p : (action \cup object)^n \mapsto perceptible$. E.g. $isOn(block1, block2)$. For each (relational or functional) fluent P there is a perceptible isP, but there are perceptibles also for object description, e.g. $isBlock(x)$.
9. *stochastic action*.
   $\langle ., . \rangle : outcome \times action \mapsto stochasticAction$.
   E.g. $\langle v(grasp(block2)), grasp(block2) \rangle$.
   With $v : action \mapsto outcome$, e.g. $v(grasp(block2)) = 0$.
10. *event*. $\circ : stochasticAction \times seqEvent \mapsto seqEvent$.
    $| : stochasticAction \times seqEvent \mapsto condEvent$, and the special constant E_0.
    E.g. $\langle 0, grasp(block1) \rangle \circ \langle 1, goTo(table, block2) \rangle \circ E_0$ is a sequence of stochastic actions, i.e. is a term of sort $seqEvent$, hence a term of sort event.
11. *measure*. $\mu_P : object^n \times seqEvent \mapsto object$.
    E.g. $\mu_{On}(block1, block2, E_0) = 0.7$. For each fluent F there is a belief measure μ_F.
12. *probability* **P**: $stochasticAction \times event \mapsto object$.
    E.g. $\mathbf{P}(\langle 0, grasp(block1) \rangle | E_0) = 0.9$.
13. $\prec$: $seqEvent \times seqEvent$. E.g. $E_0 \prec \langle v(a), a \rangle \circ E_0$.
14. *Could*. $stochasticAction \times event$, e.g. $Could(\langle 1, grasp(block1) \rangle, S_0)$.
15. *Percept*. $(perceptible \times outcome \times situation)$.
    E.g. $Percept(isClear(block1), 1, do(sense(isClear(block1, 1), S_0)))$.
16. *sit*: $event \mapsto situation$,
    e.g. $sit(\langle 1, fall(block1) \rangle \circ event) = do(fall(block1, S_0))$.

3. THE BASIC THEORY OF ACTIONS

The notions related to a basic theory of action with its set of foundational axioms and domain description axioms, are presented in detail in [19] and in [17]. These axioms represent the robot's prior knowledge about its domain of application, and are denoted by the following:

$$\mathcal{D}_{S_0} \cup \mathcal{D}_{ssa} \cup \mathcal{D}_{una} \cup \mathcal{D}_{ap} \cup \Sigma.$$

Here, $\mathcal{D}_{S_0}$ is the initial database, describing what properties hold initially about the objects in the domain. $\mathcal{D}_{ssa} \cup \mathcal{D}_{ap}$ specify, respectively, the successor state axioms and the action precondition axioms. The successor state axioms provide the causal laws governing the domain together with a solution to the frame problem. Σ is the set of foundational axioms for situations, and $\mathcal{D}_{una}$ ensures the uniqueness of each action. The above sets specify the basic ontology of the application domain and the robot's features.

3.1 The basic ontology for the blocks world

Here we present an example, the blocks world, which is ArmHandØ's domain of application. ArmHandØ is just one of the three manipulators, each having its specific ontology, we cannot describe all of them due to the limited space. First note that to deal with the real world domain the position of each object in the scene needs to be known to the robot, possibly acquired through perception. The robot's workspace is the table, which, in turn, is divided into landmarks. Constants in the language are used to reference objects in the *real world*.

3.1.1 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 manipulator end effector. So for example in the initial database $\mathcal{D}_{S_0}$ we would have sentences of the kind:

$$\begin{array}{ll} Block(x) & \equiv ((x = rb_1 \vee \ldots \vee x = rb_n) \wedge Red(x)) \vee \\ & (((x = bb_1 \vee \ldots \vee x = bb_k) \wedge Blue(x)) \vee \ldots). \\ Land(x) & \equiv (x = lm_1 \vee \ldots \vee x = lm_k). \\ EndEf(x) & \equiv x = ee. \end{array}$$

Besides descriptions, which are not affected by world dynamics 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.

3.1.2 Fluents

$On(x, y, s)$: Object x is on the object/landmark y, in situation s. To handle the projection of the position of a block x on the map of landmarks we introduce the predicate $Bottom(x, y, s)$, recursively defined as follows:

$$\begin{aligned} Bottom(x, y, s) \;\; &\equiv On(x, y, s) \wedge Land(y) \vee \\ &\exists z.On(x, z, s) \wedge Bottom(z, y, s). \end{aligned}$$

The intuitive meaning of the above definition is: the projection of a block x is on a landmark y on the map, in situation s, iff either x is on y and y is a landmark or there is some other block in between.
$Clear(x, s)$: Object x is clear, in situation s.
$Holding(x, s)$: The end-effector holds object x, in situation s.

3.1.3 Control Actions

$moveTo(x, y)$: 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.

3.1.4 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.

3.1.5 Successor State Axioms

The role of the successor state axioms (SSA) is to provide a definition for the causal laws of the domain, integrated with a solution to the frame problem [19], i.e.*the execution of an action affects only what it has to, as specified in the SSA, and nothing else*. In particular the SSA satisfy the Markovian property that any change in a state depends only on the previous one. An example of a SSA for the basic ontology is the following:

$$\begin{aligned} &On(x, y, do(a, s)) \equiv a = moveTo(x, y) \vee a = shift(x, y) \vee \\ &\quad On(x, y, s) \wedge \\ &\quad (a \neq grasp(x) \wedge a \neq fall(x) \wedge a \neq slip \wedge \\ &\quad \forall z.(z \neq y \rightarrow (a \neq shift(x, z) \wedge a \neq moveTo(x, z)))). \end{aligned}$$

The above axiom says that a block x will be on y in a situation $do(a, s)$ in which a action a occurred, just in case such an action was either a deterministic action, like $moveTo(x, y)$ or it was an exogenous action, like $shift(x, y)$.

4. BEYOND THE BASIC ONTOLOGY

4.1 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 blocks world are $isLand, isBlock, isEndEf$ as follows:

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

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:

$$\begin{aligned} &Scene(isBlock(x), do(a, s)) \equiv a = \\ &observeScene(isBlock(rb_1), \ldots, isLand(lm_k), pr_1, \ldots, pr_n) \wedge \\ &(x = rb_1 \wedge pr_1 = 1 \vee \ldots \vee x = bb_k \wedge pr_m = 1) \wedge \\ &Descr(isBlock(x)) \vee Scene(isBlock(x), s) \wedge \\ &\forall p_1..p_n\; pr_1..pr_n.a \neq observeScene(p_1, \ldots, p_n, , pr_1, \ldots, pr_n) \end{aligned}$$

Here $Descr(isBlock(x))$ is a suitable description of a block, e.g. containing information about the color, 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:

$$\begin{aligned} &Percept(isOn(x, y), pr, s) \equiv x \neq y \wedge \\ &\quad (pr = 1 \wedge \\ &\qquad Scene(isBlock(x), s) \wedge (Scene(isBlock(y), s) \vee \\ &\quad Scene(isLand(y), s)) \wedge Descr(isOn(x, y))) \vee \\ &\quad (pr = 0 \wedge \\ &\quad \neg(Scene(isBlock(x), s) \wedge (Scene(isBlock(y), s) \vee \\ &\quad Scene(isLand(y), s)) \wedge Descr(isOn(x, y)))). \end{aligned}$$

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 we might get, for example, $On(a, b, s)$ and $Percept(isOn(a, b), 0, s)$. These discrepancies are recorded by a defined predicate, i.e. not in $\mathcal{L}_{SC}$, $Mistaken(p, s)$, 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].

4.2 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 1. 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.

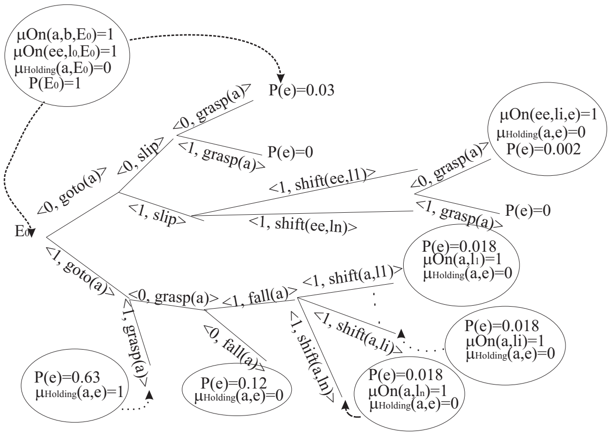

Figure 1: The tree $tree(do(goTo(a), do(grasp(a), S_0)))$

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.

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 $[0, 1]$.

4.2.1 Axioms for events

We extend the axioms for situations to events as follows, with $\# \in \{\circ, |\}$ and all free variable are implicitly universally quantified:

1. $\neg(e \prec E_0)$
2. $sta \circ e \neq sta|e$
3. $(\langle v(a), a\rangle \# e) = (\langle v(a'), a'\rangle \# e') \equiv a = a' \wedge e = e'$
4. $e \prec (sta \# e') \equiv e \prec e' \vee e = e'$
5. $P(E_0) \wedge \forall\, sta.P(sta \# E_0) \wedge$
   $\forall sta' \forall e.P(sta \# e) \rightarrow P(sta \# (sta' \circ e)) \rightarrow$
   $\forall sta \forall e\; P(sta \# e)$
6. $(P(E_0) \wedge P(sta \circ e) \rightarrow \forall w P(w)) \wedge$
   $(P(sta|\, E_0) \wedge P(sta|e) \rightarrow \forall u P(u)) \wedge$
   $\forall w \forall u.P(w) \wedge P(u) \equiv \forall e P(e)$

The above axioms are a suitable counterpart for events, of the axioms for situation. In fact, these axioms impose a tree-structure for events (see Figure 1) in any interpretation; moreover they ensure that any property P of events has a standard model, that is, one in which there is no strange object holding properties of events. Note that for all events of sort sequence the analogous theorems as for situations [17] are derivable.

4.2.2 Axioms for probability **P**

1. $\mathbf{P}(e) \geq 0$, for any event e.
2. $\mathbf{P}(E_0) = 1$.
3. $\mathbf{P}(sta \circ e) = \mathbf{P}(sta \mid e) \times \mathbf{P}(e)$
4. $\mathbf{P}(\langle 0, a\rangle | \, e) = 1 - P(\langle 1, a\rangle \mid e)$.

Let $\nu \in \{0, 1\}$. We define the following functions:

$$sit(E_0) = S_0$$
$$sit(\langle \nu, a\rangle \circ e) = do(a, sit(e))$$
$$outcome(\langle \nu, a\rangle \circ e, \langle \nu, a\rangle) = \nu.$$

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

$$\begin{aligned} tree(s) &= \{\langle \nu, a\rangle \circ w \mid ExAct(a) \wedge w \in tree(s) \vee \\ &\qquad sit(\langle \nu, a\rangle \circ w) \sqsubseteq s\} \\ cut(s) &= \{w \mid w \in tree(s) \wedge \forall w'.w' \in cut \rightarrow w \not\prec w'\} \\ leaves(s) &= \{w \mid w \in tree(s) \wedge \neg\exists w'.w' \in tree(s) \wedge w \preceq w'\} \end{aligned}$$

Given the above axioms and the above functions, the following properties hold:

1. $\mathbf{P}(sta|e) = \dfrac{\mathbf{P}(sta \circ e)}{\mathbf{P}(e)}$
2. $\mathbf{P}(\langle 1, a\rangle \, | e) + \mathbf{P}(\langle 0, a\rangle \mid e) = 1$
3. Let $leaves(\sigma) = \{w_1, \ldots, w_k\}$ then $\sum_i \mathbf{P}(w_i) = 1$.
4. Let $\{w_1, \ldots, w_k\}$ be a cut of $tree(\sigma)$ then $\sum_i \mathbf{P}(w_i) = 1$.

4.3 Belief measures of fluents

To represent the properties holding at a stochastic sequence of actions, that is, at a supposed event, we introduce the notion of measure. For each fluent $F(\vec{t}, s)$ we add a functional event fluent $\mu_F(\vec{t}, e)$, with the same arity as F and such that all arguments which are not of sort situation are the same as F and the argument of sort situation is replaced by an argument of sort event. The initial database axiomatization manages the coherence of the measures with the fluents.
Given a sequence of stochastic actions $w = \langle v(\alpha_1), \alpha_1\rangle \circ \ldots \circ \langle v(\alpha_n), \alpha_n\rangle \circ E_0$, 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:

$$Could(\langle 1, a\rangle, w) \equiv \Psi_{\langle 1,a\rangle}(1, a, w)$$

where Ψ specifies the conditions for executing the action a with outcome 1. The above axiom states which preconditions, in terms of values of belief measures of fluents 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 landmark 3 and it slipped, it follows that the measure of the fluent $On(ee, lm_3)$ cannot be 1 and the action $grasp(a)$, 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:

$$\mathbf{P}(\langle 1, a\rangle \mid e) = p \equiv \Phi(sta, e) \tag{1}$$

Here, $\Phi(a, e)$ is a formula with free variables a, e defining the conditions, given e, of success of the stochastic action sta. We show here how we define these axioms for the blocks world, in which initial information might have been acquired through perception. Consider the control and exogenous actions given in paragraph *3.1.3* and *3.1.4*.

$$grasp(x), goTo(x), moveTo(x, y), fall(x), slip, shift(x, y)$$

Likewise, consider the measures of belief associated with the fluents given in paragraph *3.1.2*.

$$\mu_{On}(x, y, e), \mu_{Holding}(x, e), \mu_{Clear}(x, e), \mu_{Bottom}(x, y, e)$$

For example, the preconditions for $grasp$ and for the exogenous action $fall$ are:

$$\begin{aligned} &Could(\langle 1, grasp(x)\rangle, e) \equiv Block(x) \wedge \mu_{Clear}(x, e) = 1 \wedge \\ &\quad \mu_{On}(ee, x, e) = 1 \wedge \forall z.Block(z) \rightarrow \mu_{Holding}(z, e) = 0. \\ &Could(\langle 1, fall(x)\rangle, sta \circ e) \equiv sta = \langle 0, grasp(x)\rangle \vee \\ &\quad \exists y.sta = \langle 0, moveTo(x, y)\rangle. \end{aligned}$$

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 $goTo$ or the action $moveTo$ just failed. Given the above preconditions, the conditional probabilities for non exogenous actions can now be defined, as follows:

$$\begin{aligned}&\mathbf{P}(\langle 1, grasp(x)\rangle| \, e) = p \equiv Could(\langle 1, grasp(x)\rangle, e) \wedge p = 0.7 \vee \\ &\quad \neg Could(\langle 1, grasp(x)\rangle, e) \wedge p = 0.\end{aligned}$$

As we have seen above, the preconditions for the success of an exogenous action like $fall(x)$ depends on the result of a previous action -because of the completeness failure assumption. For example the failure of closing the hand can determine that the block would fall on the table. 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 $fall(x)$, conditioned on the failure of a stochastic action, and the conditional probability of $shift(x, y)$ conditioned on the success of the exogenous actions $fall$ and $slip$:

$$\begin{aligned}&\mathbf{P}(\langle 1, fall(x)\rangle \mid sta \circ e) = p \equiv Could(\langle 1, fall(x)\rangle, sta \circ e) \wedge \\ &\quad (sta = \langle 0, grasp(x)\rangle \wedge p = 0.6) \vee \\ &\quad (sta = \langle 0, moveTo(x, y)\rangle \wedge p = 0.4) \vee \\ &\quad p = 0 \wedge \neg Could(\langle 1, fall(x)\rangle, sta \circ e).\end{aligned}$$

$$\begin{aligned}&\mathbf{P}(\langle 1, shift(x, y)\rangle \mid sta \circ e) = p \equiv \\ &Could(\langle 1, shift(x, y)\rangle, sta \circ e) \wedge \exists z.\mu_{Bottom}(x, z, e) = 1 \wedge \\ &(sta = \langle 1, fall(x)\rangle \wedge p = Prob_{sh}(z, y) \times \mathbf{P}(\langle 1, fall(x)\rangle | e)) \vee \\ &(sta = \langle 1, slip\rangle \wedge p = Prob_{sh}(z, y) \times \mathbf{P}(\langle 1, slip\rangle | e)) \vee \\ &\quad p = 0 \wedge \neg Could(\langle 1, shift(x, y)\rangle, sta \circ e).\end{aligned}$$

Here, $Prob_{sh}(lm_i, lm_j)$ is a macro defined, in the initial database, for each pair of landmarks and it is independent of events and situations and just gives the probability distribution of the shifting of an object, after a fall or a slide, from one position to another.
Observe that by the axioms for **P** we have for each of the above stochastic actions also a conditional probability for failure:

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

Successor state axioms for fluent measures. The successor state axioms for fluent measures define how a measure changes depending on the occurred event. We present the measures for $\mu_{Holding}$ and μ_{Clear}:

$$\begin{aligned}&\mu_{Holding}(x, sta \circ e) = p \equiv sta = \langle v(a), a\rangle \wedge \\ &\quad (p = v(a) \wedge a = grasp(x)) \vee \\ &\quad (p = 1 - v(a) \wedge \mu_{holding}(x, e) = 1 \wedge \\ &\quad a = fall(x) \vee \exists z.a = moveTo(x, z)).\end{aligned}$$

$$\begin{aligned}&\mu_{Clear}(x, sta \circ e) = p \equiv (sta = \langle v(a), a\rangle \wedge \\ &\quad p = v(a) \wedge \exists z.\mu_{On}(z, x, e) = 1 \wedge \\ &\quad a = grasp(x) \vee \exists y.a = moveTo(z, y) \vee a = fall(z)) \vee \\ &\quad (p = 1 \wedge \mu_{Clear}(x, e) = 1 \wedge \\ &\quad \forall z.(z \neq x \rightarrow sta \neq \langle 1, moveTo(z, x)\rangle \vee \\ &\quad (Land(x) \rightarrow sta \neq \langle 1, shift(z, x)\rangle)).\end{aligned}$$

4.4 Diagnosis of failures

At the end of the execution of a task the action $observeScene$ 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 $Percept(isOn(b, a), 1, s)$, 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 $moveTo(b, lm3)$ but it failed to pick up b then the agent database would entail $On(b, lm3)$, 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.

4.4.1 Likelihood

Let σ be a sequence of actions. Let $w = sta_1 \circ \cdots \circ sta_k \circ E_0 \in leaf(tree(\sigma))$ be a final sequence of stochastic actions.

$$\begin{aligned}&likelihood(w) = 1 \equiv_{def} \mathbf{P}(w) \neq 0 \wedge \\ &\quad \forall x\, y.Block(x) \wedge (Block(y) \vee Land(y)) \rightarrow \\ &\quad Percept(isOn(x, y), 1, \sigma) \rightarrow (\mu_{On}(x, y, w) = 1 \wedge \\ &\quad \forall x.Percept(isHolding(x), pr, \sigma) \rightarrow \mu_{Holding}(x, w) = pr \wedge \\ &\quad Percept(isClear(x), 1, \sigma) \rightarrow \mu_{Clear}(x, w) = 1.\end{aligned}$$

We define, now, a preference relation $<_\gamma$ between stochastic sequences as follows:

$$w <_\gamma w' \equiv_{def} likelihood(w) < likelihood(w') \wedge \mathbf{P}(w) < \mathbf{P}(w').$$

Note that $<_\gamma$ induces a partial order. Consider the set of $<_\gamma$-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 than in w'. These preferences are reasonable and, in general, enough to identify a unique preferred sequence w of stochastic actions in the set of $<_\gamma$-maximal elements.
Once the most likely sequence w is computed, according to a preference criterion γ, 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 $w_{max_\gamma} \in tree(\sigma)$ be the sequence of stochastic actions satisfying the maximum likelihood hypothesis according to the preference criterion and the heuristic. Let $\sigma' = sit(w_{max_\gamma})$. There exists a sequence $\sigma^\star$, which can be obtained from σ', such that $\sigma^\star$ *explains* the perception in σ:

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

$$\begin{aligned}&\mathcal{D} \models sit(w) = \sigma \wedge \sigma^\star = trans(w) \wedge \\ &\quad \forall \vec{x}\; Percept(isP(\vec{x}), 1, \sigma) \rightarrow P(\vec{x}, \sigma^\star)\end{aligned}$$

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

5. IMPLEMENTATION

5.1 Perception

The role of perception is to correctly instantiate the sensing action $observeScene$ and to update the local map. We model the scene with a hierarchical geometric model whose components are the connections among the entities in the scene and their properties. The entities are: $cubes$, $color$, $endEf$, $gride$ and a special object ($unknown$) for anything else. 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*. An alignment procedure will extract all the needed information about the geometric transformations, like rotation, translation and scaling.

The unknown entities, instead, are represented with a bounding-box surrounding the entity. The scene, in turn, is a tree rooted at the grid node, with entities as nodes and a topological relation between entities as edges. The root node may have one edge for each region. The other nodes in the structure represent the cubes with features, the end effector with features or unknown objects. This intermediate node may have only one edge. More specifically this structure is described using the following notation:

$$
\begin{aligned}
&scene = \{LandList, EntityList\}\\
&EntityList = \{entity_1, entity_2, \ldots\}\\
&LandList = \{landmark_1, landmark_2, ...\}\\
&entity_i = \{[\langle cube, feature\rangle | \langle EndEf, feature\rangle | unknown],\\
&\qquad\qquad geometricTrasformation\}\\
&landmark_j = \{1, 3, 7, \ldots\}
\end{aligned}
$$

where each landmark description is a sequence of integers indexing the entities list. In figure 2, an example of this structure (the pyramid represents the end effector) is shown for the blocks world.

The action $observeScene$ amounts to visiting the above tree-

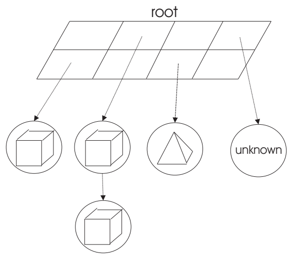

Figure 2: The scene representation tree.

structure. For example, the branch $landmark_3 = \{1, 3\}$ with $entity_1 = \{\langle cube, red, \phi_1\rangle\}$ and $entity_3 = \{\langle cube, blu, \phi_3\rangle\}$, will instantiate the action $observeScene$ as follows:

$$
\begin{aligned}
&observeScene(isBlock(rb_1), isBlock(bb_3), \ldots,\\
&\qquad isLand(lm_3), \ldots, 1, 1 \ldots, 1)
\end{aligned}
$$

In turn the above instantiation affects also the instantiation of the perceptible $isOn(rb_1, bb_3)$. In particular the 3D structure of the environment is obtained by combining the information contained into two separate images. After the interpretation of the image the map, which is a simple description of the position of the entities in the workspace (see the table below), is updated.

$cube_i$	$color_i$	x_i	y_i	z_i	ϕ_i
$EndEffector$	$null$	x_{ee}	y_{ee}	z_{ee}	ϕ_{ee}
$unknown$	$null$	x_j	y_j	z_j	ϕ_j
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

In the table above ϕ_i represent the orientation of the instances of the entities with respect to the reference frame.

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.

5.2 High Level Control

The high level control is managed by a Golog programs. Golog [11] 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 : $\alpha : \beta$
2. Test action : $\phi?$
3. Nondeterministic *choice* : $\alpha|\beta$
4. Nondeterministic choice of arguments : $(\pi x)\alpha$
5. While loop : α^*
6. Loops and Procedures, including recursion.

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

$$
\begin{aligned}
&primtive\ action : Do(\alpha, s, s') \stackrel{def}{=} poss(a, s) \wedge s = do(a, s)\\
&test : Do(\phi?, s, s') \stackrel{def}{=} \phi[s] \wedge s = s'\\
&sequence : Do(\alpha : \beta, s, s') \stackrel{def}{=} \exists s^* Do(\alpha, s, s^*) \wedge Do(\beta, s^*, s')\\
&nondet.\ choice : Do(a|b, s, s') \stackrel{def}{=} Do(a, s, s') \vee Do(b, s, s')\\
&nondet.\ choice\ of\ arg. : Do((\pi x)\alpha, s, s') \stackrel{def}{=} \exists x Do(\alpha, s, s')
\end{aligned}
$$

Both procedure calls and blocks with local procedure declarations can be defined; for details see [11].

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:

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

where $sit(e)$ and $outcome(sta \circ e, sta)$ are the functions defined in Section *4.2.1*. 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.

$$
DoG(prog, s, s') \stackrel{def}{=} Do(\delta, s, s') \wedge exec(s, s')
$$

where

$$
exec(s, s') \equiv \forall s''\ a.s \sqsubseteq do(a, s'') \sqsubseteq s' \rightarrow PossG(a, s'')
$$

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 δ is *low level executable*. $PossG$ is then a sort of guarded $Poss$, where $findPath$ represents a guard for the action preconditions. Such a guard verifies 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 $chooseTask$,
  $goal|[(\neg goal)?; \pi(x)(task(x); observeScene)]$
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 in the blocks world.

proc $task(1),$
$\pi(x)(\pi(l)((clear(x) \wedge \neg onTable(x,l) \wedge$
$\neg goodTower(x))?)\ ;\ goTo(x)\ ;\ grasp(x)\ ;$
$\pi(l_1)(moveTo(x,l_1); land(l_1)?)\ ;\ goTo(l_0))$
endProc
proc $task(2),$
$\pi(x)(\ \pi(y)(\ (clear(x) \wedge clear(y) \wedge$
$onTable(x) \wedge goodTower(y) \wedge sameColor(x,y))?\ ;$
$goTo(x)\ ;\ grasp(x)\ ;\ moveTo(x,y)\ ;\ goTo(l0)))$
endProc
proc $task(rec),$
$\pi(x)(\ (object(x) \wedge badPosition(x))?;$
$goTo(x)\ ;\ grasp(x)\ ; \pi(y)(\ (land(y) \wedge$
$\neg badPosition(y))?\ ; moveTo(x,y)))$
endProc

$onTable(x,s)$ is an abbreviation for: $\exists l(land(l) \wedge on(x,l))$.
$task(1)$ 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 it 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 $generateEvent(s),$
$(s = s_0)?|\pi(s')(\pi(a)((s = do(a,s'))?;$
$generateEvent(s'); stocasticActions(a)))$
endProc
proc $stocasticAction(a), \langle 1,a\rangle|\langle 0,a\rangle; compAction$
endProc
proc $compAction,$
$\pi(x)(exogenousAction(x)?:\langle 1,a\rangle|\langle 0,a\rangle)$
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 $DoG(\delta, s, s')$ 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.

5.3 Local Control

The Local Planner takes care of the effective executability of a task and of the physical actions affecting the motors. It controls the robot movements between Cartesian coordinates. Its purpose is to find both a collision-free path for the manipulator and the configurations of each joint to move from one location to another. An initial map of the manipulator workspace is supplied to the local planner. This map is updated by the visual map at the end of each task, that is, by the geometric map supplied by perception. Given a sequence of actions, e.g $\sigma = do(grasp(a), do(goTo(a), S_0))$, the *local preconditions* of each actions are evaluated by the guarded $PossG$ and the function $findPath(a)$ is computed by the Local Planner, where a refers to a's center of gravity coordinates, in the current local map. To evaluate these functions, first some intermediate positions are found, between the actual frame of the manipulator and the frame of a, using a dynamic version of the $A^\star$ algorithm [20, 15], and then for each intermediate position, a series of joint angles are computed to move the joints through until the tool frame has reached the subgoal frame. If a failure is returned, then the sequence is not possible and the search restarts for the task. Otherwise, the local map is updated by the perceptual map obtained from the instantiation of $observeScene$, as described in 5.1. To compute the entire manipulator motion between two configurations two different but complementary modules have been implemented: one for the direct kinematics problem and one for the inverse kinematics problem. The chain link model of the manipulator yields the so-called Denavit-Hartenberg parameters. Using these parameters, we get the system of equations describing exactly the direct kinematics of the robot. To invert this system we have been using a closed form solution method [4]; the equations we get are the four different solutions (one for each joint) to the inverse kinematics problem. These equations constitute a planner that works in a one-shot fashion. According to the analytic solution obtained of the direct and inverse kinematics of the manipulator, the angular values are then used by the motor-control actions $moveJoint(\#, \theta)$.

6. 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 is 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 framework, 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. This makes, in general, a considerable difference because states, for us, are only implicitly defined. Probabilities in [18] are used to represent one's ignorance about the initial state – we actually could also have this – and the outcomes of actions.

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 [13, 14] 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 similar approach has been taken by Grosskreutz and Lakemeyer (see [8, 7]). They mainly focus on the low level processes representation and projection, while our aim is to extend the high level representation with a model of failures, perception and diagnosis. In particular,in [8] they present a probabilistic model to capture the non deterministic nature of low level processes (pGOLOG) affecting high level actions (mGOLOG). Instead, we model directly high level actions failure extending the Situation Calculus with a new notion of event. The difference is mainly ontological: we model high level failures while [8] model low level processes. A high level programming approach is a central issue also for Funge [5] 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 [6]). 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.

7. 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).

8. 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] J. Craig. *Introduction to Robotics, Mechanics and Control*. Addison-Wesley Pub. Co., 1986.

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

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

[7] H. Grosskreutz and G. Lakemeyer. cc-golog: Towards more realistic logic-based robot controllers. In *AAAI/IAAI*, pages 476–482, 2000.

[8] H. Grosskreutz and G. Lakemeyer. Turning highlevel plans into robot programs in uncertain domains. In Submitted. http://www-i5.informatik.rwth-aachenbonn.de/kbsg/publications/., 2000.

[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] 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.

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

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

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

[15] N. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, 1971.

[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] A. Stentz. The focused D* algorithm for real-time replanning. In *Proceeding of IJCAI*, pages 1652–1659, 1995.