A system integrating high and low level planning with a 3D-visualizer

Alberto Finzi, Fiora Pirri, Marco Schaerf
Dipartimento di Informatica e Sistemistica
via Salaria 113, 00198 Roma, ITALy
e-mail{finzi,pirri,schaerf}@dis.uniroma1.it

Abstract

One of the most difficult problem in the designing of planning systems concerns the integration of several reasoning components like sensing, perception and high level planning together with robotics modeling techniques. In this paper we present a simulation system for a robotic arm operating on a platform of the ISS (JERICO domain). We have defined a hierarchy of planners at the task, global and local level that suitably interact to account for different levels of control of the execution of the tasks. The task planner, realized in GOLOG, utilizes a KB in the Situation Calculus to find the sequence of abstract actions necessary to reach the goal. The global and local planners expand each abstract action into a more refined sequence computing a path in the workspace (global planner) and then in the configuration space (local planner). We illustrate the 3-dimensional graphical interface and the robot simulation module and how it interacts with the planning system.

1 Introduction

One of the most difficult problem in designing a planning systems concerns the integration of several reasoning components like perception, scheduling, execution monitoring and planning together with manipulation planning, motion planning and sensing. As McDermott and Hendler have remarked in their introductory paper of the AI-journal special issue on planning [1], scaling a planning domain usually yields a set of problems that involve a lot of reasoning techniques from other fields. In this paper we present a proposal for decomposing a manipulation planning problem into a hierarchy of planners and integrate them with a 3D-visualizer. The idea of adopting a hierarchy of models has been also investigated by Cameron [5]. The novelty of our approach relies on the fact that we use a symbolic model of the domain for the task/high level planner as opposed to the geometrical model of the workspace used for the low level planners. The advantage of a hierarchy of planners is manifold. Here we quote only two aspects: computational complexity and modularity. For the first aspect consider that the domain independent planning problem is in general undecidable; however under certain restrictions like when there are no function symbols and only finitely many constant symbols then planning is decidable and its computational complexity varies from constant time to EXPACE-complete [6]. On the other hand planning a manipulation path to bring the movable objects to their specified goal location is PSPACE-hard [21]. Now, for planning, most of the complexity is to be found in the way preconditions for actions are formally represented while for manipulation – and in general motion – planning it is to be found in the geometrical representation of the workspace and in its dimension. Keeping the geometrical model of the workspace separated from the symbolic model of the domain is thus necessary to avoid an increase in complexity on both the planners. Another advantage of the hierarchy is modularity. Local planners depend on a particular robot, on its linkages, joints, degree of freedom, while global and task planners can be both formulated independently from the specific structure of the manipulator. Therefore, under a suitable decomposition, the same task and global planners can be adapted to several manipulators.

The paper is organized as follows. In the next section we introduce some preliminaries just to specify the notation. Then we present the hierarchical planner decomposed into task, global and local planners and we discuss some example in the literature that use an analogous hierarchy. In Section 4 we introduce the symbolic model formalized in the Situation Calculus and the way primitive and complex actions are managed via the axiomatization and the programming language GOLOG. In Section 5 we discuss the geometrical model of the workspace in which all objects are assumed to be convex. We then introduce the global planner and its interaction with both the task and the local planner. In Section 6 we introduce the local level problem, the module taking care of the robot kinematics in the configuration space together with the graphic module that simulates the robot executing the complex tasks required to achieve a goal. Finally we address some further issues that we have not included in this presentation.

Proc. Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space, 1-3 June 1999 (ESA SP-440)

2 Preliminaries

In most part of the paper we refer to the Situation Calculus [16, 8], a first order language with sorts. The three disjoint sorts are: *action* for actions, *situation* for situations and *object* for everything else depending on the domain of application. We refer the reader to the literature, e.g. [17], for a detailed presentation of the alphabet of $\mathcal{L}_{sitcalc}$ and for the metalanguage adopted to denote terms and formulae of the language. The alphabet includes relations and functions called Fluents, because their truth value depends on the history of actions performed by an agent: a history, like

$$[grasp(payload), rotate(payload),\\ ungrasp(payload), rotate(handle)..]$$

is designated by a situation s. A situation s is the last argument of Fluents, e.g $Handle(payload5, do(pickUp(payload5, S_0))$, where $s = do(pickUp(payload5, S_0)$. The Situation Calculus is a powerful basic axiomatization for representing dynamic domains. We define a *basic theory of actions* to be a set of axioms describing the preconditions for each action that can be dealt with by an agent,via the *Action Precondition Axioms*, the effect of each action via the *Successor State Axioms* and the initial situation that we call D_{S_0}, in which no action, relative to the current task, has been executed.

Example 1 Consider the following fluents:

$inContainer(p, n, s)$:
payload p is in nest n in situation s
$locked(n, s)$: nest n is locked in s
$position(pos, s)$:
end-effector is in position pos in s
$holding(obj, s)$:
end-effector is holding the obj in s
$posPlat(pos, s)$: position of the mobile platform

And the following *primitive actions*:

$goto(pos)$:	move to the *pos* position
$lock(nest)$:	lock the *nest*
$unlock(nest)$:	unlock the *nest*
$extract(nest)$:	extract the object (if it exists) contained in *nest*
$insert(nest)$:	insert the object (if the arms is holding something) in *nest*
$move(plat, \theta)$:	move the mobile platform to the specified orientation θ

An initial situation S_0 can be defined in this way:

$$locked(nest_1, S_0)\\ \neg\exists p(holding(p, S_0))$$

Actions precondition axioms for each primitive action have the form:

$$Poss(lock(nest), s) \equiv\\ position(handle(nest), s) \wedge \neg\exists x(holding(x, s)) \wedge\\ \neg locked(nest, s)$$

Successor State Axioms for each fluent have the form:

$$locked(nest, do(a, s)) \equiv a = lock(nest) \vee\\ locked(nest, s) \wedge a \neq unlock(nest)$$

Complex actions can be dealt with via the programming language GOLOG [15] (alGOL in LOGic), whose declarative semantics is given in the Situation Calculus. GOLOG is a logic-programming language which, in addition to the primitive actions axiomatized as specified above, allows the definition of complex actions using programming constructs which are like those known from conventional programming languages like conditionals, iteration, procedures.

What is special about GOLOG is that the meaning of these constructs is completely defined by sentences in the Situation Calculus. For this purpose, a macro $Do(\rho, s, s')$ is introduced whose intuitive meaning is that executing the program ρ in situation s leads to situation s'. Here we provide some sample definition needed for Do. See [15] for the complete list.

$Do(A, s, s') \doteq Poss(A, s) \wedge s' = do(A, s)$, where A is a primitive action.

$Do(\textbf{if } \varphi \textbf{ then } \rho_1 \textbf{ else } \rho_2 \textbf{ endif}, s, s') \doteq$
$Do([(\varphi?;\ \rho_1)|(\neg\varphi?;\ \rho_2)], s, s')$

Here φ is a formula of the Situation Calculus with all situation arguments suppressed.

For specific sections of the paper we assume the reader familiar with basic robotics terminology, we refer to [10] for a full introduction. We recall that a *Configuration* is a mathematical specification of the position and orientation of every body composing a robot, relative to a coordinate system. The configuration space C is the set of all configurations of a robot. The configuration space has dimension m, where m is the number of the degree of freedom (dofs). The number of dofs of a robot arm is equal to its number of joints. We denote by C_{obj} the configuration space of the object Obj.

3 The Hierarchy

Consider a manipulator, a 6 or 7 degree of freedom robot arm working on a platform where there are payloads installed in nests and locked. An action like *pickUp(payload5)* can be considered a primitive one, at the level of abstraction at which we are used to think about *simple actions*. On the other hand, picking up the payload may require a huge amount of simpler actions like verifying whether the payload is really reachable, where it is, and thus moving to the payload position – avoiding all the obstacles – unlocking the handle of the nest where the payload is installed, rotating the end-effector once or twice so as to rotate the payload for detaching it from the nest and finally pulling the payload out of the nest.

Now, we are still missing something: each of the more detailed actions in which *pickUp* has been decomposed actually refers only to the end effector. In fact,

we have to consider the whole arm, which is a collection of bodies, connected by joints, having constraints. These constraints have to be satisfied in the space of all the configurations that the robot arm can assume, while *passively* following the end-effector. What we have just described is the simplest and natural hierarchy that a manipulation planning problem requires. The hierarchy we are proposing decomposes the planning problem into different abstraction levels allowing to manage challenging domains both from a conceptual (logical) point of view and from the geometrical and dynamical ones. Examples of three layered architectures can be found in [13], where successful mobots, developed with the P-SA approach, are described. Interesting examples are *RHINO* [18] and *Saphira* [14]. The upper level of these systems requires a Task Planner but the lower levels are usually reactively managed. This is possible because the robots considered are involved in tasks in which low level behavior can be driven on-line. In general, however, for manipulation planning where robots arms are involved, off-line planning at each level is required.

Task level We consider an autonomous agent potentially able to achieve any complex task like delivering hot coffee in an office, cooking pasta and serving it, moving any number of blocks on a table so as to form any sophisticated shape [17]. All these complex tasks are potentially achievable as far as we are concerned with a symbolic model of the world, taking care of the causal laws governing preconditions and postconditions of each primitive action, together with a suitable solution to the frame problem [16]. A solution to the frame problem specifies what in the domain has been changed and what remains unchanged after the execution of an action. Tasks and goals at this level are formalized using a domain theory and a basic theory of actions (see Section 2 above) that provides, for each primitive action that can be executed, i.e. whose preconditions are satisfied, a full description and formal characterization. Complex actions can be obtained by composing primitive actions in the programming language GOLOG.

At the Task level the preconditions to any control action already en-globe a solution to the problems of controlling the real forces applied to the end effector, of finding a free space for the path needed to execute the action, and of a transfer path for correctly manipulating the objects. In other words any action executed at the task level *can* be executed because all the space problems have *already* been solved. As the formalization relies on this assumption, at the task level each primitive action can be considered as an idealized representation of the physical world and the agent as a free-flying object.

The role of the task planner is to give the agent the postulates to reason about the domain and to coordinate her actions in an intelligent behaviour so as to achieve the required goals.

Example 2 Consider the domain JERICO (Joint European In-Orbit Calibration and Operation) defined for the Russian segment of the International Space Station, see Figure 1. Given an initial configuration of the payloads, locked in their nests, and an initial configuration of the exchange terminal and the pointing platform, the agent – the robotic arm – has to reorient the pointing platform and move the payloads from their nests to other nests by turning the nest-handle to unlock the payload, transferring them to the required nest, eventually using the exchange terminal, turning the payload to insert it in the nest and finally locking the handle. Observe that to remove or install a payload into a nest the agent has to ungrasp the handle and regrasp it such that it can be rotated.

Figure 1: Jerico domain

The Task planner provides us with very interesting off-line plans that can also take into account sensing and perception.

Global level In Latombe [12] a manipulation path is defined as an alternating sequence of *transit* and *transfer* paths that connect an initial configuration q_{init} to a goal configuration q_{goal}. A *transit* path is an arm's motion that does not move any object. A *transfer* path defines an arm's motion that does move an object. In our hierarchical model, actions are executed only at the task and global level, therefore transit and transfer paths are defined at the global level. The global level is formalized within a geometrical model of the agent workspace in which both the agent and objects are assumed to be convex. Objects in the workspace, including the robot end effector are represented within particular bounding volumes called cylspheres [9], that is, cylinders with semi-spheres of the same ray of the cylinder added on top, see Figure 2.
The global level takes care of computing a manipulation path – free from collision – for the end effector from an initial situation S_0^g to a final situation S_{goal}^g that satisfies the postconditions of a given task action a. In other words, given a task level action a, the global level *expands* such a single action into a sequence of manipulation actions $[ma_1, \ldots ma_n]$ that satisfy the geometrical constraints of the workspace, that is, avoids the obstacles and correctly manipulate the movable objects.

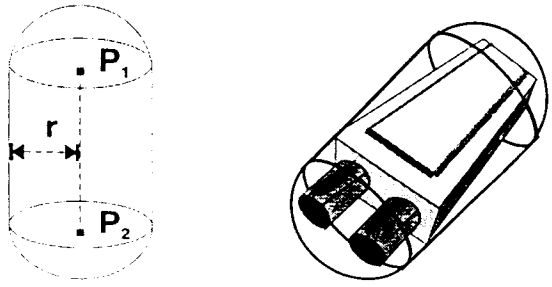

Figure 2: A cylsphere bounding the end effector.

The role of the global level is to ensure that all the preconditions required to execute action a – within the workspace – are satisfied. Our strategy uses Latombe idea [11] consisting in representing the end effector as having a dynamic shape that changes together with the objects it is manipulating; see Figure 3.

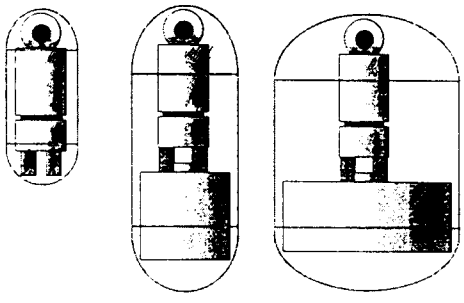

Figure 3: The end effector shape depends on the object is manipulating.

We formalize the geometrical model also in the Situation Calculus.

Example 3 Suppose the task level has delivered a sequence of actions $[a_1, \ldots, a_m]$ and its present situation is s. Suppose also that at situation s the action $a = pickUp(payload5)$ has to be executed, the task level queries the global level to verify whether the geometrical preconditions for a to be executed are satisfied. The current situation s is transformed into the global initial situation S_0^g, which is the start situation for the global planner. A sequence of actions $[ma_1, \ldots, ma_k]$ is then computed by the global planner and are such that the situation s^g reached by the execution of these actions satisfies the postconditions of $pickUp(payload5)$.

Local level The *local planning* step manages the whole structure of the arm, namely its end-effector, elbow, joints etc. The local planner makes a constrained search in order to achieve a safe path for each joint. The movements to which the arm is committed are strictly dependent on local information.

Planning at this level can be done in several ways. A variable that influences the local planner architecture is the grid step used in the global planning phase. In particular, if this step is small then a *ONE-SHOT planner* it is needed [11] that reaches directly a final position for the robot avoiding collisions. Otherwise, if the step is large, a more powerful planner [3, 5] is necessary that produces intermediate configurations for the arm.

Our local planner belongs to the category of *ONE-SHOT planners* that are based on inverse kinematics algorithms. These algorithms iteratively calculate a final configuration starting from an initial one and a final position for the end-effector. The module that calculates the inverse kinematics is the same used by the simulator.

Discussion Our architecture is close to the idea described in [5], although our Task level is far different from the one proposed by Cameron as we use a symbolic model formalized in the Situation Calculus. Cameron' s view is to split between tactical knowledge (Task and Global levels) and geometric (Global and Local levels). We agree that the Cameron's structure has several advantages: the decomposition came natural in solving the manipulation and planning problem, the planner is easier to understand and to modify and eventually to adapt to a new domain or to upgrade it. Our system has developed the Task layer, that in the Cameron system is considered as a marginal aspect of the architecture, and its connection with the Global level (consistency between the two representation and communication between the two modules). In addition our Global Planner is developed as an interface between the logical representation and the geometrical one. Our representation of the word is mixed: the metrical representation is connected to a Knowledge Base that allows, when necessary, to perform some spatial reasoning. At the local level the system proposed by Cameron uses an approach based upon *virtual forces* whilst we have used a kinematic approach. The Global-Local interaction therefore is similar to the one described in [3] where Local planning consists in special inverse kinematic algorithm and Global planning is developed using RPP.

4 The Task planner

At the task level, we define a basic theory of actions representing the virtual attitude of the agent to reason about the domain. A sequence $[a_1, \ldots, a_n]$ of actions that the agent executes at this level leads the agent into a situation s. In s the domain has been transformed by the actions executed by the agent. The transformation is witnessed by the truth values of the fluents. When the language is suitably restricted, the set of fluents $\langle F_1(s), \ldots, F_n(s) \rangle$ which are entailed by the basic ac-

tion theory, at situation s, is a state that will be used to interact with the global and local levels.

Following Green [7], given a set of conditions on the domain that has to be satisfied and which we call a *Goal*, a *plan* is any sequence of actions $[a_1, \ldots, a_m]$, whose preconditions are satisfied and are such that in the situation $s = [a_1, \ldots, a_m]$ the *Goal* is verified. Formally, if $\mathcal{D}$ is a basic theory of actions, as we defined in the preliminaries, and *Goal* is a set of conditions, we require that:

$$\mathcal{D} \models \exists s.\ Executable(s) \wedge Goal(s)$$

In the Situation Calculus, since the preconditions for each action are suitably axiomatized the above definition implies that s is a plan whenever $\exists s Goal(s)$ is a theorem of the basic theory of actions. In particular, given an initial domain specification $\mathcal{D}_{S_0}$, if $\mathcal{D}_{S_0}$ is a complete theory about the initial situation, it is always possible to determine whether there exists a situation $s = [a_1, \ldots, a_n]$ in which the *Goal* is satisfied and such a situation, if it exists, can be constructively given via any sound and complete deductive method.

The axiomatization of the situation calculus ensures that the search space is a tree rooted in S_0. Starting from the initial situation, the Task Planner searches for a sequence of actions that leads to a situation where the goal is satisfied. The search is driven by an heuristic that can be well defined using the expressiveness of the language: the heuristic is described in the Situation Calculus as well, using the fluents introduced for the basic theory of action, that express both the knowledge and the meta-knowledge.

Following the approach of [2] we use a domain specific knowledge to control the search of a forward chaining planner. To this end we have introduced two special fluents: $badSituation(s)$ [17] and $sugg(a, s')$ that indicates respectively: a situation s in which it is not useful to search the goal and the action a that is suggested in the situation s'. For example:

$$badSituation(do(goto(x), do(goto(y), s)$$

The above statement cuts out all situations in which the end-effector moves toward a position and then moves away without accomplishing any task in the situation where she arrives. On the other hand a suggestion can be defined as follows:

$$sugg(insert(payload(y), nest(x)), s) \leftarrow goodInCont(payload(y), nest(x))$$

here, *goodInCont* is a predicate that is true iff the payload y must be in the nest x in the final configuration. This formula suggests to insert a payload in a nest (when it is possible) that must contain that payload in a final configuration.

An interesting property of this planner is that it represents a compromise between deductive planning and planning as a search process. The Task Planner is implemented as a GOLOG procedure that searches for a plan in the space of situations. If a sequence $[a_1, \ldots, a_n]$ satisfies the *Goal* then it is accepted as the plan.

The Knowledge Base can be easily implemented as a PROLOG program. In the case of a complete representation of the domain, it is possible to exploit PROLOG as a theorem prover (in this case negation as failure is valid), otherwise (see Open Word Golog in [17]) it is necessary to use a theorem prover ad hoc developed for domains written in the Situation Calculus. The GOLOG interpreter is written in Prolog as well [17].

Developing a Task Planner in GOLOG has several advantages. With this language it is possible to exploit properties like: quick prototyping, expressiveness of the KB, integration between knowledge and meta-knowledge, integration between procedural and denotational way of programming using automated reasoning just when it is strictly necessary. These features are very important: GOLOG is a procedural language that can directly use the Knowledge Base to deliberate when it is needed. In this way, during the execution of the program, it is possible to access the Knowledge Base testing the validity of some property, but also to control the execution by explicit meta-level knowledge (in our case the heuristics defined by the fluents $badSituation(s)$ and $sugg(a, s)$). Therefore with our GOLOG planner the trade-off between expressiveness of the Knowledge Base and computational complexity of the planning task is addressed finding a way between writing a high level control program (that is the classical GOLOG approach [15]) and developing a backward search planner.

5 The global planner and the geometric domain

Objects and the end-effectors are represented by particular bounding-volumes called cylsphere [9]. A cylsphere is just the 3D projection of a segment and its geometrical structure is defined by the centers of the two semi-spheres and by the ray common to the cylinder. A basic volume of this kind is well specified using two points and a ray. As we observed above we represent the end effector as a cylsphere of varying dimension, depending on the payload carried in the transfer part of the manipulation.

The distance between two cylsphere can be reduced to the distance between two segments. The distance point-segment is defined along the perpendicular to the line, to which the segment belongs, passing through the

point P.

$$M = P_1 + At$$

with $A = (P_2 - P_1)/||P_2 - P_1||$ the verse of the line and P_1, P_2 the extreme points of the segment. Once the parameter t is known, the distance d is:

$$d = ||P - M||$$

The distance between two segments is always defined on the perpendicular to the line to which the segment belongs but one has to take care of problems like complanarity and parallelism. The distance between two cylspheres is defined accordingly. In fact it results from the distance between the segments defined by the extremes of the solids to which the value of the rays has to be subtracted. A function *Bbox* applied to any object in the geometrical model will give back the bounding volume of the object as a cylsphere. The geometrical model is also axiomatized in the Situation Calculus, but the domain objects denotes only the reals. To capture the relations between objects we define a hierarchy similar to the one adopted for graphical applications. Each object is represented using two parameters: the distance between vertices and the ray and a transformation function involving the ancestor nodes in the hierarchy. A functional fluent $Edge(x, y, s^g)$ represents the geometric transformation of x w.r.t. y in the geometrical situation s^g.

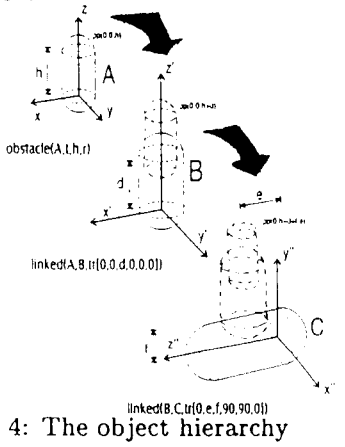

Figure 4: The object hierarchy

Movable objects are simply linked to the nests frames, according to the current task state. A subset of the objects that could be considered as obstacles for the manipulator is defined as a state, that is, at a given situation s each object is described in terms of the coordinates of its cylsphere; see Figure 4). The exploration starts from the node specified and goes back towards the root applying all the transformation encountered to the cylinder-sphere contained in the starting node.

Global planning means searching for a manipulation path of the end effector as if it will be free from the rest of the body. The global planner generates a sequence of wrist positions and orientations so that the end-effector shall avoid obstacles and reach the final task.

The search algorithm proposed is a special translation of A^* driven by a heuristic that minimize the straight-line distance between the current state and the goal one. The expansion step takes care of the current arrangement of the end-effector and of the payload carried represented with their own cylinder-sphere, and avoids all states that generate a collision in the environment.

Each orientation of the end-effector comes from an interpolation between the initial and the final orientation desired, re-calculated at each iteration of the A^* algorithm. See Figure 5).

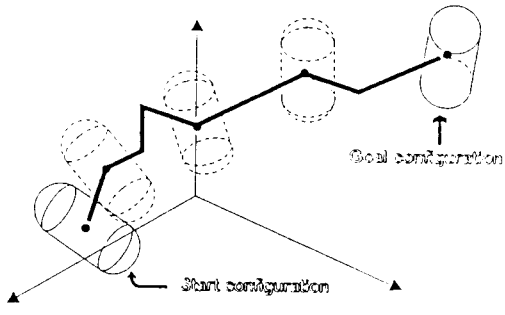

Figure 5:

To improve efficiency, we introduce a grid in the 3D space whose step-size is determined by the complexity of the world where the manipulator acts. A large step decreases the number of moves needed to reach a goal state, but could generate a non collision-free path because it doesn't take care of obstacles that lie between two adjacent positions. On the other hand a small step increases the resolution, but also the number of steps required to reach the goal. For this reason we define a variable step that can be defined at the beginning of the computation according to the complexity of the environment. The global planner has been implemented in Prolog.

6 Local Planner

The role of the local planner is to verify and refine the manipulation plan delivered by the global planner. At this point some paths may by found to be impossible; in such a case the global planner has to find alternative solutions otherwise the task planner has to re-plan.

The local planning problem, in our hierarchical structure, is defined as follows. A configuration C^s_{obj} is generated by the state of the global planner defined at situation s. A state is the vector $< p_1(s) \ldots p_n(s) >$ of all positions and orientations of the objects in the work space at the situation s^g, that is the current geometrical situation of the global planner. The configuration C^s_{e-f} is the subset of the work space occupied by the end-effector and generated by the state

$< q_1(s) \ldots q_m(s) >$, where $q_1(s) \ldots q_n(s)$ are the positions and orientations of the cylindersphere bounding the end-effector and eventually the object it is manipulating.

Given C^s_{obj} and C^s_{e-f} and a sequence of actions $[ma_1 \ldots ma_n]$ executable at the global level and a sequence of states associated with situations $s_1 \ldots s_n$, the problem is to find configuration spaces $C_1 \ldots C_n$, where C_i is the set of all configurations of the robot arm and objects at the state $< p_1(s) \ldots p_n(s), q_1(s) \ldots q_m(s) >$, such that there exists a collision free path for the whole arm for executing $[ma_1 \ldots ma_n]$. Observe that since $[ma_1 \ldots ma_n]$ is a coarse expansion of a task action a, in order to find a manipulation sequence, collision free, for the end effector, any subset of $[ma_1 \ldots ma_n]$ satisfying the preconditions and postconditions or any sequence of configurations for a, at the geometrical model, would be accepted. To solve the local planning problem we have used an inverse kinematics algorithm based on the computation of the transpose of the Jacobian matrix [19].

The method relies on the linear relationship between end-effector and joint velocities; it was early introduced by Wolovich and Elliot [20]. Sciavicco Siciliano in [19] applied the method to redundant manipulators and showed that the redundant degrees of freedom could be used to satisfy both obstacle avoidance constraints, and constraints on joint ranges of motion.

The method works as follows. Considering a composite force F applied to the end-effector, this external force will result in internal torques and forces at the joints. The relation between F and the internal forces τ can be written as:

$$\tau = J^T F \tag{1}$$

This suggests an iterative method for forcing the end-effector to track a time-varying trajectory $x_d(t)$. If the current end-effector position is $x_c(t)$, then the error measure.

$$e(t) = x_d(t) - x_c(t) \tag{2}$$

can be thought of as a force f pulling the end-effector toward the desired trajectory point $x_d(t)$. From this force we can calculate the joints velocities q'.

$$q' = J^T F \tag{3}$$

A single integration step yields a new vector q which moves the end-effector towards $x_d(t)$. This procedure repeats until the end-effector reaches the desired position, or some other stopping condition is met.

The method ensures that only forward kinematic calculation is required and in general problems with matrix singularities are avoided. Their occurrence can be overcome using an integration method with an adaptative step-size.

The local planner has been implemented in JAVA.

7 The graphic Interface

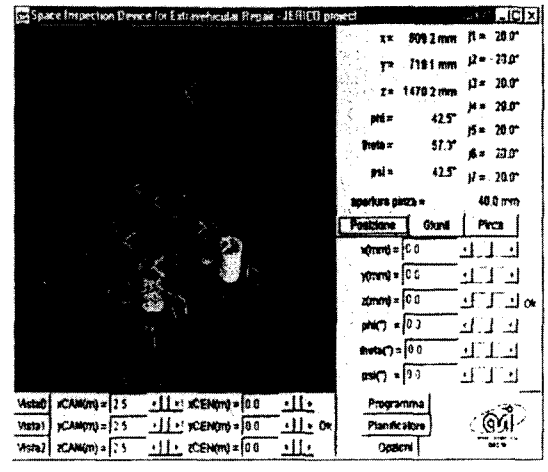

Figure 6: Graphical interface

The three-dimensional user interface allows to manage in a visual and friendly way the operations required by the robot arm during the execution of the tasks. In particular the user can perform the following operations.

1. Observe the evolution of the scene in a window visualizing the 3-D animation of both the robot and the object it is manipulating.
2. Observe the state of the robot on some panels in which there are information about end-effector position and orientation, joints angles, etc.
3. Manually interact with the robot arm specifying a final configuration for the arm: joints angles, end-effector position, hand opening etc.
4. Writing a program to accomplish a specific task.
5. Graphically define a final scene configuration in terms of payload dispositions in nests, or nests and platform orientations.

The actual scene is represented in a window and the final one is obtained modifying this scene by the mouse. The final configuration activates the planner that produces a program directly executed by the robot simulator

8 Conclusions

We have presented a modular decomposition of a planning system for a manipulator. We have integrated our system together with a simulator and a graphical interface. We have developed the system with conditional plans and perception. The modularity allows to take care of both perception and spatial reasoning. We are now concerned with the run-time behaviour of the planner and with execution monitoring.

Acknowledgments

This project has been found by ASI (Italian Space Agency). We thanks Ray Reiter, Andrea Pompili, Giuseppe Rossi and Angelo Dell'Arciprete for their help and important contributions to the realization of this work.

References

[1] D. McDermott, J. Hendler *Planning: What it is, What it could be, An introduction to the Special Issue on Planning and Scheduling. Artificial Intelligence Journal* vol. 76(1-2), pages 1-16,1995.

[2] Bacchus and Kabanza. *Using Temporal Logic to Control Search in a Forward Chaining Planner. New Directions in Planning, M.Ghallab and A.Milani(Eds.)10S Press* pages 141-153,1996.

[3] J.Barraquand, J.C.Latombe. *Robot Motions Planning: A Distruibuited Rappresentation Approach Int. J.Rob.Research,10(6),1991.*

[4] A. McLean, S.Cameron. *Effective Path Planning and Collision Avoidance for Redundant Manipulator.* Int. Conf. Adv. Rob. & Comp. Vision,1996.

[5] S.Cameron. *Dealing with Geometric Complexity in Motion Planning..* IEEE Conf. Rob. & Automaton,1996.

[6] K. Erol, D.S. Nau, V.S. Subrahmanian. 1995 *Complexity, decidability and undecidability results for domain-independent planning. Artificial Intelligence Journal* vol. 76(1-2), pages 74-88,1995.

[7] C.C.Green. Theorem Proving by resolution as basis for question-answering systems. In B.Meltzer and D.Michie, editors, *Machine Intelligence 4*, pages 183-205. American Elsevier, New york, 1969.

[8] P.J.Hayes and J.McCarthy.1969. *Some Philosophical Problems from the Standpoint of Artificial Intelligence. Machine Intelligence 4.* B.Melzer and D.Michie eds., Edinburgh University Press 463-502.

[9] D. Henrich, X. Cheng. Fast Distance Computation for on-line Collision detection with multi-arm robots. In *IEEE International Conference on Robotics and Automation* pp. 2514-2519, 1992.

[10] J.C. Latombe. *Robot motion planning* Kluwer Academic Publishers, 1991.

[11] Y. Koga, K. Kondo, J. Kuffner,J.C. Latombe. *Planning Motions with Intentions* In *Proc. SIGGRAPH'94,394-408,1994.*

[12] Y. Koga and J.C. Latombe. *On Multi-arm manipulation planning* In *International Conference on Robotics and Automation.* pp 945-952, 1994.

[13] D. Kortenkamp, R. Bonasso and R. Murphy eds. *AI-based Mobile Robots: Case studies of successful robot systems* Cambridge,MA:MIT Press.1998

[14] K. Konolige and K. Meyer. The Saphira architecture for autonomous mobile robots. In *AI-based Mobile Robots: Case studies of successful robot systems* Cambridge,MA:MIT Press.1998

[15] Y. Lesperance, H.J. Levesque, F. Lin, R. Reiter and R.B. Scherl. GOLOG: a logic programming language for dynamic domains. *Journal of Logic Programming*,31,59-84,1997.

[16] R.Reiter. *The Frame Problem in the Situation Calculus: A simple solution (sometimes) and a completeness result for goal regression. Artificial Intelligence and Mathematical theory of Computation:papers in Honor of John McCarthy,*Vladimir Lifschitz (ed.),Academic Press,San Diego,CA,1991,pp.359-380.

[17] R. Reiter.*Knowledge in Action: Logical Foundations for Describing and Implementing Dynamic Systems.* MIT Press, to appear. *http://www.cs.toronto.edu/cogrobo/*

[18] Burgard, W., Cremers, A. B., Fox, D., Hähnel, D., Lakemeyer, G., Schulz, D., Steiner, W., Thrun, S., The Interactive Museum Tour-Guide Robot, *AAAI-98.*

[19] L. Sciavicco and B. Siciliano. A dynamic solution to inverse kinematic problem of redundant manipulators. In *IEEE Int.Conference on Robotics and Automotion*, pages 1081-1086, March 1987.

[20] W.A. Wolovich and H. Elliot. A computational technique for inverse kinematics problem of mechanical manipulators. *IEEE Transactions on Robotics and Automation*, 1988

[21] G. Wilfong. Motion Planning in the presence of movable obstacles. *Proceedings of the 4th ACM Symposium on Computational Geometry.* 1988, pp 279-288.