Reactive Planning Simulation in Dynamic Environments with *VirtualRobot*

Oscar Sapena[1], Eva Onaindía[2], Martín Mellado[3], Carlos Correcher[3] *, and Eduardo Vendrell[3]

[1] Dept. de Ciencias de la Computación e Inteligencia Artificial, Univ. Alicante, Spain
osapena@dccia.ua.es
[2] Dept. Sistemas Informáticos y Computación, Univ. Politécnica Valencia, Spain
onaindia@dsic.upv.es
[3] Dept. Ingeniería de Sistemas y Automática, Univ. Politécnica Valencia, Spain
martin@isa.upv.es, carlos.correcher@jrc.it, even@isa.upv.es

Abstract. This paper describes the architecture of a reactive planning system for dynamic environments, which is specifically designed to deal with robot planning problems. The architecture permits many agents to work simultaneously on the same environment and it is aimed at working with incomplete information. Agents have partial knowledge about the world and data soon becomes obsolete because of the changes in the environment. Our approach is designed to overcome this difficulty through a highly coupled system composed of an incremental planner and an executor. The whole system is integrated into *VirtualRobot*, a graphical software application, which provides a flexible and open platform to work on robotics. Through *VirtualRobot* we can incorporate important features into the system as simulation of sensing actions or a monitoring mechanism. Additionally, the planning algorithm is able to work in time-limited situations and use numeric variables. All these features make our planning system be a nice toolkit to deal with reactive robot planning.

Introduction

The problem of classical planning involves generating a sequence of actions which, applied to an initial state, allows to achieve a set of goals. Research in classical planning has been carried out under some non-realistic assumptions as static, deterministic and completely accessible environments [12]. In order to overcome these simplifications, new approaches like universal, conformant, conditional or probabilistic planning have arisen. Nevertheless, in these approaches, the plan execution monitoring is just a simple mechanism that executes the received plan and checks that everything happens as planned. In case that an unexpected event occurs, it is necessary to compute a new plan from scratch or to try to repair the old plan [2].

* Currently working in Institute for the Protection and Security of the Citizen (IPSC), Joint Research Centre – ISPRA.

In general, plan generation for autonomous systems (like mobile robots) cannot be separated from its execution. Planned actions will be executed in unpredictable and dynamic environments, so plans are more likely to fail. Moreover, there is information that can only be acquired during execution time. Reactive planning integrates plan generation and execution, which constitutes a suitable platform to deal with real-world problems. In reactive planning, an agent is defined as a combination of a planner plus a reactor [13], and this approach is the basis of the work presented in this paper.

The integration of the planner and the reactor is carried out through the use of *VirtualRobot Simulator* (*VRS)*, a graphical simulator part of *VirtualRobot* suite. Nowadays, graphic simulators for robotic systems are indispensable in most of the robot design, learning and exploitation steps. Technological advances and improvements in computing engineering allow this kind of applications to be applied on any field in robotics: industrial robotics [3], mobile robotics [14], sub-aquatic robotics [1] or aerospatial robotics [7]. In addition, they have become significantly more powerful and flexible. Thanks to the inclusion of new capabilities such as sensor data, simulation software is not only dedicated to simulate robot behaviour, but also is useful for design, analysis and validation of techniques as collision detection, motion planning, sensor modelling, evaluation of control architectures and so on.

VirtualRobot has been originally created for remote robot monitoring, programming and simulation of the robot control system *GENERIS*[*] [11], but has become a useful general tool in many fields of robotics, such as manipulator-robot programming, walking robot simulation, mobile-robot control, distance computation, sensor simulation, collision detection, motion planning and so on.

System architecture

The system is composed of several agents working in the same environment. Agents can be classified in planning agents and external agents. Agents of the former type are composed of a planner and a reactor (see Fig. 1). They are in charge of computing and executing plans to achieve goals, which can vary along the time. External agents, like people or natural phenomena, are capable of modifying the environment without the need of the knowledge of the planning agents. They produce most of the unexpected events that the planning agent detects during the plan execution through a monitoring process.

Initially, planning agents have some knowledge about the state of the world (environment model). Normally, this knowledge will be incomplete and inaccurate. The reactor asks the planner for one or more actions when it is required in the domain. If there are no deadlines to return a plan, the answer can be delayed until the planner has computed a good plan. When the reactor requests an action, the planner must reply as soon as possible according to the environment time constraints. The reactor translates the action into a set of low-level actions (see Example 1) and it sends these actions to the environment. Therefore, a high-level action cannot be considered as an atomic ex-

[*] Generalised Software Control System for Industrial Robots, developed by European Commission Joint Research Centre

ecutable action, so the reactor must have a recovery mechanism to reach a valid state if the action execution fails in an intermediate stage. In this case, the reactor updates the information the planner has about the world and notifies the unexpected outcome to the planner.

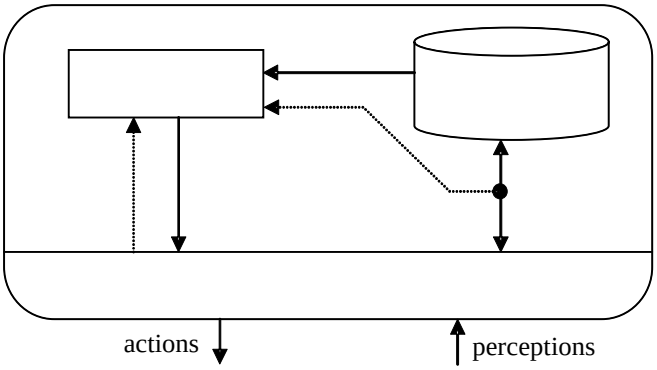

Fig. 1. A planning agent is composed of a planner and a reactor.

Example 1. One high-level action (`MOVE rob A B`) for moving a differential robot `rob` from room `A` to room `B` can be translated into the following sequence of low-level actions:

```
compute_angle α between A and B;
if (battery_available) then rotate rob α degrees;
while (battery_available) and (not collision) and
      (position(rob) ≠ B) do one_step_forward rob;
end while;
```

The graphical simulation of the execution is carried out through the *VRS* application. At initialization time the system input are three parameters: a domain description, a problem description and the environment description. The system is able to work under any robotics graphical simulator so in the environment description it must be specified that *VRS* is the toolkit to be used. The integration with *VRS* is done through a dynamic link to a library. The library has to receive the following information during its initialization:

- The domain name.
- The environment description. This description is provided in a separate file, which contains information about the geometry, colors and textures of the static objects (objects that cannot be moved or altered) in the environment.

– The problem objects: each object has a name, a type, some coordinates to locate the object in the environment and a physical model (containing geometric and kinematics parameters). *VRS* distinguishes two types of objects: parts - objects that can be handled - and crafts - objects that can modify the environment.

Once the environment is initialized, the system creates the agents. Planning agents receive the domain and problem description as well as the library to communicate with the environment. The planner uses the information from the problem and domain to compute the plans. The reactor, which is domain-dependent, is implemented into the library. The communication between the reactor and the planner is established through two callback functions:

– ***Environment_action_request***: this function is used by the reactor to request the planner for an action.
– ***Environment_monitor_info***: this function is used by the reactor to inform the planner about unexpected events.

When the planner receives an action request, the process of generating plans is halted and the planner sends an action to the reactor through the function ***SendAction***. The reactor translates this high-level action into primitive actions, which are directly executable in the environment and are understandable by *VRS*. If the action execution fails, the reactor must be able to reach a high-level state, i.e. a valid planning state. Then, the reactor communicates the planner the unexpected changes in the environment. If these changes affect the calculated plans, the planner restarts the computation of new plans and discards the old ones (for the moment, the planner does not use any replanning technique to reuse the old plans).

Reactive planning

In a reactive environment it is not feasible to find a complete plan before starting the execution. That is the reason why the planning algorithm follows the design principles of the anytime algorithms [2]. The planning algorithm is also based on the divide-and-conquer methodology: split the problem into smaller subproblems, solve these subproblems and combine the obtained solutions. The planning algorithm computes a plan for each top-level goal separately. However, combining these plans may cause conflicts between each other that are hard to solve [15].

The overall working scheme is shown in Fig. 2. A planning problem `P=(O,I,G)` is a triple where `O` is the set of operators, `I` the initial state and `G` the top-level goals. This algorithm starts from the current state S_0, which initially corresponds to `I`. The planning algorithm works in four stages:

– **The preprocess stage**. This stage consists of computing all ground actions and literals, starting from the operators, predicates and objects. These calculations speed up the following planning stages. This preprocessing stage is usually done

once but, in dynamic environments, the number of actions and literals can change, so new objects can appear giving rise to new possible actions and literals.

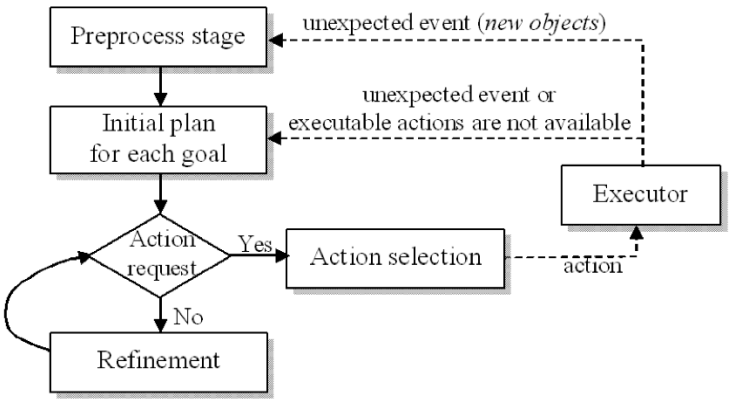

Fig. 2. Outline of the planning algorithm.

- **Calculation of the initial plans**. An incomplete plan is computed for each non-achieved goal $g_i \;/\; g_i \in G \;\wedge\; g_i \notin S_0$. Therefore, P is decomposed in n planning subproblems $P_1=(O,S_0,g_1)$, $P_2=(O,S_0,g_2)$,..., $P_n=(O,S_0,g_n)$, where n is the number of non-achieved goals. These plans are computed starting from a relaxed planning graph (where delete effects of actions are ignored) [4], and they are generated very rapidly (polynomial time). We must remark that these plans are not completely executable in most cases. However, these initial plans are aimed to be used as a starting point for further refinements.
- **Refinement of the plans**. While the reactor does not require actions to be executed, the planner proceeds with the refinement of the incomplete plans. This refinement continues until all plans are completed, an unexpected event invalidates the current plans or the available time expires. The refinement stage repairs the plans through the insertion of new actions to make the plan completely executable. Each refinement stage attempts to solve a non-solved literal in each plan.
- **Selection of the action to be executed**. When the reactor requests for an action, plans are ordered according to a conflict checking criteria [12]. This way, a plan P_i is ordered before a plan P_j when it is necessary to execute the first action of P_i before starting the execution of P_j. The next action to be executed will be the initial action of the plan ordered in the first place.

Simulation Platform: *VirtualRobot*

VirtualRobot [8][9] is a freeware software suite* in the sense that includes several programs for robotics application, research and education, with a graphical represen-

* *VirtualRobot* can be downloaded in http://www.isa.upv.es/~vrs

tation based on *OpenGL*. *VirtualRobot* is designed for low cost platforms and used as the common interface for all the applications. *VirtualRobot* is composed of:

- A basic geometric modeller *VirtualRobot Modeller* (*VRM*), to create and edit geometric and kinematics models.
- A geometric data translator, *VirtualRobot Translator* (*VRT*) to convert files from *AutoCAD® DXF* and *VRML* file formats in addition to a special plug-in module to export data from *3DStudio Max®*.
- The main platform for simulation, *VirtualRobot Simulator* (*VRS*), including some components (a set of adaptable Dynamic Link Libraries) and external applications (*VRS Tools*, *VRM Tools* and *VRS Demos*).

VRS can be applied for simulating any type of robots, individually or grouped in multi-robot workcells for their off-line program generation and testing, as well as for on-line programming and monitoring. *VRS* can be connected to the numerical control *GENERIS* using *TCP/IP* to monitor the process and robots status. *VRS* can simulate any kind of robots, that is, not only manipulator-robots, but also multi-axis machines as conveyors, turntables, machine-tools, sensor systems and crafts (general mobile-robots). Robots can be attached one to one, or many to one, in order to form more complex and redundant devices, such as walking robots.

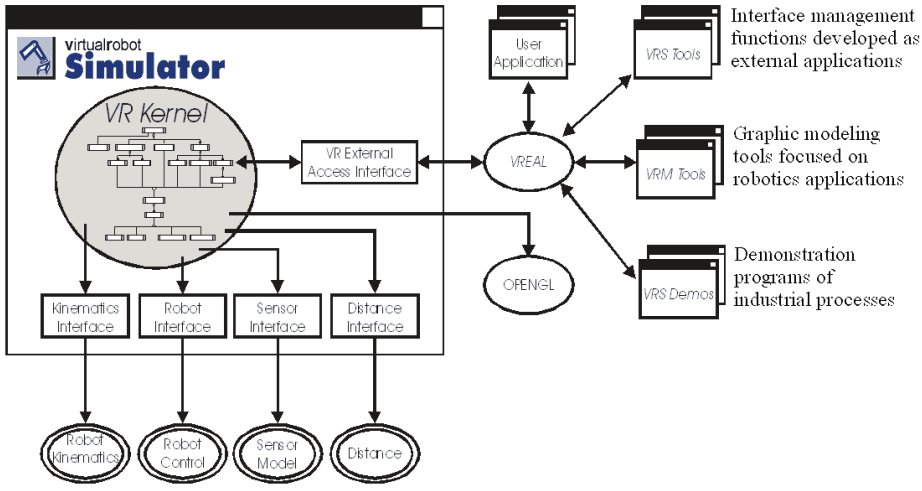

Fig. 3. *VRS* Software architecture.

VRS software architecture

VRS software architecture (Fig. 3) is based on the following four parts:

1) *VRS Kernel*, including graphical display control, objects data designed through a hierarchical structure of classes following the inclusion principle and processing threads. *VRS Kernel* handles user action events and external application orders.

2) A set of external components under *VRS* that the user can replace with his/her own components in such a way that *VRS* can be adapted to the user requirements. The external components are Dynamic Link Libraries (*dlls*) loaded on memory during execution time. Robot kinematics, sensor models and distance computation are example of external components.
3) An external access library (*VReal)* for external application development. *VReal* is implemented as a Dynamic Link Library in order to make possible the interaction among client's application and *VRS*.
4) The external applications that run over *VRS*. The external applications form the real user interface of any specific application. Once again, the user can adapt *VRS* to special requirements, constructing the required interface as new external applications.

Planning Domain Models in *VirtualRobot*

With the help of *VirtualRobot*, typical planning domains can be modelled in order to simulate the behaviour of planning techniques. We can consider the following three representative planning domains:

- **Robot-part domains**: in this type of domains, manipulator-robots grasp and move objects, that is manipulate movable objects. Typical robot-part domains are the *Blocksworld* domain [6] and *Hanoi* domain [6]. In *VRS* it is possible to load manipulator-robots from a large library of robots and new robots can also be modelled. Parts can also be easily modelled in *VRM* specifying its geometry and grasping frames. *VRS* features for robot programming include object-oriented orders in such a way that picking and placing parts can be done in a very intuitive way.
- **Mobile-robot domains**: in this type of domains, mobile-robots navigate around their environments, searching for a goal, while avoiding obstacles (Fig. 4). Problems of transportation, like the classic planning domains *Logistics* [6] (for trucks), *Ferry* [6] (for ships) or *Satellite* [7] (for aero-spatial vehicles) are examples of mobile-robot domains. A library of characteristic crafts and their corresponding kinematics components are included within *VirtualRobot* but new crafts can be easily implemented. External applications have access through speed commands to the craft speed control in order to guide craft motion. Robot navigation in mazes or part dispatching are representative tasks in these domains. For navigation, crafts can include distance and reflector sensors.
- **Mix domains**: this type of domains combines manipulator-robots and mobile-robots to be applied on complex tasks. A typical planning domain of this kind is *Depots* [7] (with trucks and hoists). In *VRS*, robots can be attached in order to form complex devices (Fig. 5). Coordination between different robots is possible by means of different functions that establish digital and analog connections between robots.

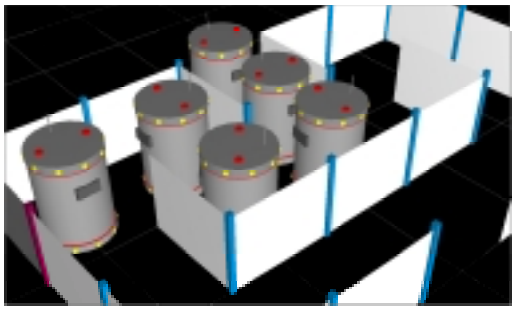

Fig. 4. Simulation of a differential wheeled robot in *VRS*.

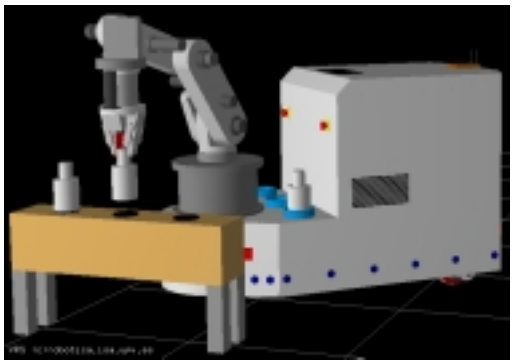

Fig. 5. A mix domain in *VRS* with a manipulator-robot attached to a tricycle wheeled robot.

Results

The standard way of evaluating the performance of planners is by comparison with other state-of-the-art planners. In classical planning, comparisons are easy since the world is assumed to be static, deterministic and accessible (closed world assumption). When the world does not comply with these simplifications, comparisons between planners are not trivial tasks. The planning community is now conscious of the new demands and, there exist several proposals to deal with more realistic assumptions. In order to do an assessment of the performance of on-line planners, they are executed with simulated scenarios. Planning quality is evaluated as a combination of the cost of the plan (referred to the problem metric function) and the running time.

Table 1 shows some preliminary results to give information about the quality and computational cost of the planning process. The table shows the plan length (in number of actions) and the running time (in milliseconds) for solving several different problems in the *Blocksworld* [6] and *Satellite* [7] domains. In *Blocksworld* domain problems, a robot arm must pick up and stack blocks in order to obtain one or more block piles in the right order. In the *Satellite* domain, it is necessary to set up several

instruments and to turn the satellites to the right direction in order to take photographs of some astronomical phenomena. Results show that the planner is able to compute plans very rapidly. Moreover, the computational cost depends on the length of the final plan rather than the problem complexity (number of objects, literals and actions), although both facts are usually interrelated. The plan length is compared to the best solution found by the planners in the competitions, since none of the planners can guarantee the optimal plan . In general, the quality of the obtained plans are very close to the best available solutions.

Table 1. Preliminary results obtained. The planner has been executed in a *K7 – 1.4 Ghz*.

	P1	P2	P3	P4	P5	P6	P7
Blocksworld	5 blocks	7 blocks	9 blocks	11 blocks	13 blocks	15 blocks	17 blocks
Plan length/Best	12/12	20/20	30/30	34/32	42/42	48/40	54/46
Time (ms.)	5	10	10	20	30	60	110
Satellite	1 sat.	2 sat.	3 sat.	4 sat.	5 sat.	6 sat.	10 sat.
Plan length/Best	9/9	13/11	20/16	26/26	35/32	42/41	48/48
Time (ms.)	5	5	40	180	350	1000	1460

Conclusions and future work

In this paper we have described a system for planning in dynamic environments and with incomplete information. The architecture of the system allows many agents to work simultaneously on the same environment and, therefore, it is possible to check the behavior of the planning agents when unexpected events occur. Moreover, the system includes some other useful features to work in real-world domains: support for sensing actions, execution monitoring, planning in time-limited situations and use of numeric variables. The environment is graphically simulated with the *VirtualRobot* toolkit, which has shown its great possibilities for this kind of problems. The architecture and integration of *VRS* with the planner have been described in the paper.

The planning algorithm is based on the divide-and-conquer methodology and computes a plan for each top-level goal separately. This technique allows to tackle quite large problems without an excessive computational effort. However, the planning algorithm is not complete but, in practice, some preliminary results show that the obtained plans are very close to the optimal ones.

There is still a lot of work to be done. It is necessary to develop simulations for some of the well-known classical planning domains, and find some new domains that can take advantage of the new system features (there are very few benchmark suites that include sensing actions and uncertainty). The system can also be improved to provide the reactor with several parallel actions in order to exploit the inherent parallelism of the real world (for example, when handling several robots at the same time). Another interesting work would be the extension of the planning algorithm to support probabilistic domains. Probabilistic domains provide the planner with additional information and help the planner choose the alternatives that are more likely to succeed. The feature of *VirtualRobot* allows to implement all these problems very easily.

Acknowledgements

This work has been partially funded by projects MCyT TIC2002-04146-C05-04, FEDER-CICYT DPI2001-2094-C03-03, DPI2002-04434-C04-04 and UPV 20020647, 20020681.

References

1. Chen, X., Marco, D., Smith, S., An, E., Ganesan, K., Healey, T.: 6 DOF Nonlinear AUV Simulation Toolbox. IEEE, (1997).
2. Drummond, M., Swanson, K., Bresina, J., Levinson, R.: Reaction-first Search. In Proceedings of the IJCAI-93. (1993) 1408-1414.
3. Freund, E., Rokossa, D., Rossman, J.: Intuitive Off-line Programming of Industrial Robots Using VR-Techniques. 15th ISPE/IEE Int. Conf. on CAD/CAM, Robotics and Factories of the Future. (1999).
4. Hoffman, J., Nebel, B.: The FF Planning System: Fast Planning Generation Through Heuristic Search. In JAIR. (2001) 14, 253-302.
5. International Planning Competition (2000). *http://www.cs.toronto.edu/aips2000*.
6. International Planning Competition (2002). *http://www.dur.ac.uk/d.p.long/competition.html*.
7. Kazuda, Y.: Experimental Study on the Dynamics and Control of a Space Robot with Experimental Free-Floating Robot Satellite (EFFORTS) Simulators. Advanced Robotics. (1995).
8. Mellado, M.: Simulación en Robótica Mediante *Virtual Robot*. Universidad Politécnica de Valencia. Ref.: 2003.443. (2003).
9. Mellado, M., Correcher, C., Catret, J.V., Puig, D.: *VirtualRobot*: An open general-purpose simulation tool for robotics. Eurosis International Conference ESMc. (2003).
10. Pollack, M.E., Horty, J.F.: There's More to Life than Making Plans: Plan Management in Dynamic, Multi-Agent Environments. AI Magazine. (1999) 20(4), 71-84.
11. Ruiz, E.: GENERIS: The EC- JRC Generalised Software Control System for Industrial Robots. Int. Journal of Industrial Robot. (1999) 26(1).
12. Sapena, O., Onaindia, E.: A Planning and Monitoring System for Dynamic Environments. Journal of Intelligent and Fuzzy Systems. (2002) 12(3-4), 151-162.
13. Wolverton, M., Washington, R.: Segmenting Reactions to Improve the Behavior of a Planning/Reacting Agent. In Proc. of AIPS. (1996) 245-250.
14. Yang, L., Yang, X., He, K., Guo, M., Zhang, B.: Research on Mobile Robot Simulation & Visualization. IEEE Int. Conf. on Systems, Man and Cybernetics. Piscataway. (1997).
15. Yang, Q.: Intelligent Planning. A Decomposition and Abstraction Based Approach. Spinger-Verlag. Berlin, Heidelberg, (1997).