Anytime Sensing, Planning and Action: A Practical Model for Robot Control

Shlomo Zilberstein and Stuart J. Russell

Computer Science Division
University of California
Berkeley, CA 94720 U.S.A.
shlomo, russell@cs.berkeley.edu

Abstract

Anytime algorithms, whose quality of results improves gradually as computation time increases, provide useful performance components for time-critical planning and control of robotic systems. In earlier work, we introduced a compilation scheme for optimal composition of anytime algorithms. In this paper we present an implementation of a navigation system in which an off-line compilation process and a run-time monitoring component guarantee the optimal allocation of time to the anytime modules. The crucial meta-level knowledge is kept in the *anytime library* in the form of *conditional performance profiles*. We also extend the notion of gradual improvement to sensing and plan execution. The result is an efficient, flexible control for robotic systems that exploits the tradeoff between time and quality in planning, sensing and plan execution.

1 Introduction

There is a wide gap between theory and practice in planning and control of robotic systems. Early theoretical work has concentrated on the analysis of systems with perfect sensors and effectors and with unlimited computational power. This paper outlines the implementation of a model of anytime computation that provides a more realistic theoretical foundation for robot planning and control. The model is based on anytime algorithms [Dean and Boddy, 1988; Horvitz, 1987] that introduce a new tradeoff in programming – between computation time and quality of results. This degree of freedom is especially useful when developing the performance elements and the control mechanism of a robotic system. The flexibility offered by anytime algorithms allows accurate sensing and extended planning when time is available, and coarse, fast sensing and planning under time pressure. It is based on the observation that in order to cope with complex environments in real-time, there is no need to sacrifice the ability to do precise sensing and planning. But, as time allocation becomes a degree of freedom, incremental scheduling and constant monitoring are necessary in order to guarantee the optimal operation of the robot. In [Russell and Zilberstein, 1991] we introduced a compilation scheme for optimal composition of anytime algorithms. It offered a new approach to the construction of complex real-time systems that separated the arrangement of the performance components from the optimization of their scheduling, and automated the latter task.

In this paper we use the compilation of anytime algorithms as part of a model for robot control. We have implemented a navigation system in which the scheduling of anytime algorithms is performed by a run-time monitoring component that uses performance information produced by the off-line compilation process. In order to reason efficiently at run-time about time allocation, we use *conditional performance profiles* that give a probabilistic description of the quality of the results of an algorithm as a function of run-time and input quality (or any set of input properties). This is an extension of an earlier notion of performance profile that depends deterministically on run-time only [Dean and Boddy, 1988].

Figure 1 shows the data flow between the main components of the system. Sensory input is used to update the description of the environment. This description is used both as input to the anytime planner and as one of the factors that determine the allocation of time by the monitor. The other factors are: the compiled performance profile of sensing and planning, the model of the environment, and the quality of the current best plan.

To demonstrate this model we have selected one of the fundamental problems facing any autonomous mobile robot: the capability to plan its own motion with noisy sensors. Section 2 describes the anytime sensing module. In Section 3, we describe the simulated environment in which the robot is situated and how the path planning problem is solved by an anytime abstract planner. In Section 4, we explain the compilation scheme that optimally integrates the anytime components of the system. Section 5 presents the run-time system. We conclude with a summary of the benefits of our approach.

2 Anytime sensing

A primary goal of this work has been to extend the notion of gradual improvement of quality to sensing. The supposition that sensors produce a perfect domain description, as much as the assumption of perfect planning and plan execution, constitutes a major disadvantage in any model for robot control. In our model, the presence of sensory errors is not an exception but rather the normal situation. Moreover, in order to optimally control the quality of sensing, the model includes a quantitative evaluation of its effect on the other components

In Proceedings of the 13th International Joint Conference on Artificial Intelligence, Chambery, France, 1993.

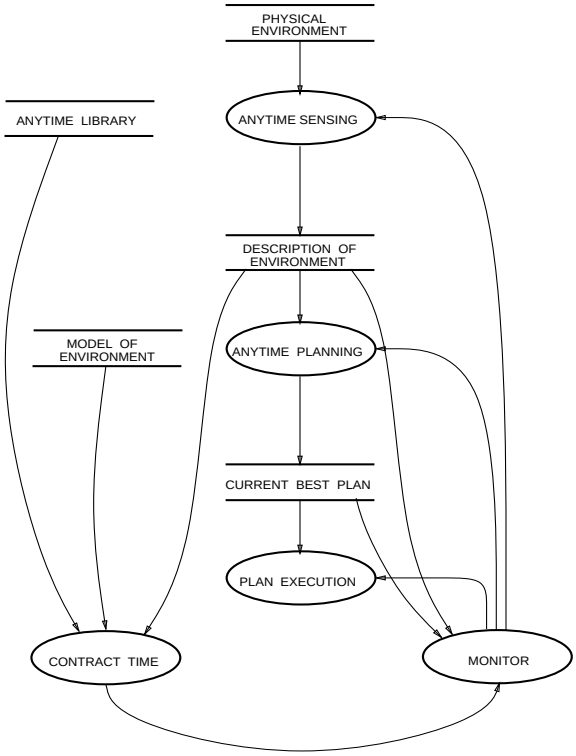


Figure 1: Data flow diagram

of the system.

This section describes the model of anytime sensing that we implemented. It produces a domain description whose quality measures the probability that an elementary (base level) position would be wrongly identified, that is, identified as free space while actually blocked by an obstacle or vice versa. We assume that within the area in which the sensors are effective, the quality of sensing is not affected by the robot's position. Our general model, however, does not require this assumption.

Figure 2 shows the performance profile of the vision module. It is characterized by several parameters: T_a, T_b, Q_a, Q_b. T_a is the minimal amount of time needed for the sensor to produce an initial domain description with quality Q_a. Given a shorter run-time, the sensor does not produce any description of the domain. For a run-time t, $T_a \leq t \leq T_b$, the quality of vision improves from Q_a to the maximal quality Q_b, which is 1.00 in this example. Detailed analysis of anytime sensing shows that in our domain sensing can be treated just as an anytime computational module for the purpose of compilation and monitoring. However, this proprty does not hold in general [Zilberstein, 1993].

3 Anytime abstract planning

Our robot is situated in a simulated, two dimensional environment with random obstacles. The robot does not have an exact map of the environment but it has a vision capability that allows it to create an approximate map. The accuracy of the domain description depends on the time allocated to the vision module. The environment is represented by a matrix

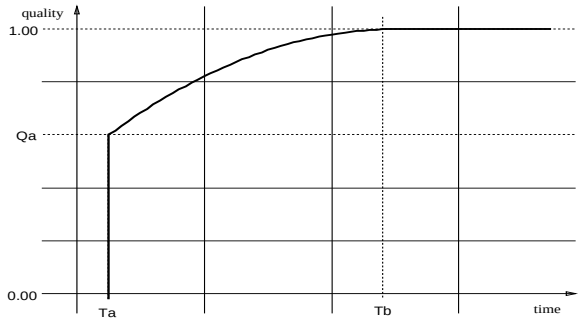


Figure 2: The performance profile of the vision module

of elementary positions. The robot can move between adjacent cells of the matrix at a varying speed which affects the execution time of the plan as well as the energy consumption. When the simulation starts, the robot is presented with a certain task that requires it to move to a particular position and perform a certain job. Associated with each task is a reward function that determines the value of the task as a function of completion time. The system is designed to control the movement of the robot, that is, determine its direction and speed at each point of time, while maximizing the overall utility. The overall utility depends on the value of the task (a time dependent function), and on the amount of energy consumed in order to complete it.

The run-time monitor has to determine at each point how much time to allocate to vision and path-planning based on factors such as the current location of the robot, the estimated distance to the goal position, the urgency of the task, and the quality of the plan produced so far.

Path planning is performed using an algorithm which is a variant of the coarse-to-fine search algorithm [Lozano-Pérez and Brooks, 1984] that allows for unresolved path segments. In order to make it an anytime algorithm, we vary the abstraction level of the domain description. This allows the algorithm to find quickly a low quality plan and then repeatedly refine it by replanning a segment of the plan in more detail. The rest of this section describes the algorithm and its performance profile.

3.1 Abstract description of the domain

In our hierarchical (quad trees) representation, the n^{th} level of abstraction corresponds to a certain coarse grid in which every position, (i, j), is an abstraction of a $2^n \times 2^n$ matrix of base-level positions. Each high level position has a certain degree of "obstacleness" associated with it which is simply the proportion of the matrix that is covered by obstacles.

3.2 The anytime planning algorithm

The interruptible anytime planner (ATP), shown in Figure 3, constructs a series of plans from *start* to *goal*, whose quality improves over time. It starts with a plan generated by performing best-first search at the highest level of abstraction (L_{max}). Then, it repeatedly refines the plan created so far by selecting the worst segment of the plan, dividing it into two segments (of identical length), and replacing each one of

```
ATP(start, goal, domain-description)
1       multi-path ← [SEGMENTIZE(start),
            PATH-FINDER(PROJECT(start, L_max),
                PROJECT(goal, L_max),
                domain-description),
            SEGMENTIZE(goal)]
2       REGISTER-RESULT(multi-path)
3       while REFINABLE(multi-path) do
4           REFINE(WORST-SEGMENT(multi-path),
                domain-description)
5           REGISTER-RESULT(multi-path)
6       SIGNAL(TERMINATION)
```

Figure 3: The anytime planning algorithm

those segments by more detailed plans at a lower abstraction level. The worst segment of the plan is selected according to the degree to which the segment is blocked by obstacles and according to its abstraction level. A special data structure, called a multi-path, is used in order to keep intermediate results. It is a list of successive path segments of arbitrary abstraction level. Note that the length of each segment of an intermediate plan is invariant. As a result, the run-time of the refinement step is approximately the same for any segment of the plan regardless of its level of abstraction.

The PATH-FINDER is a search procedure that returns the best path between any two positions in the same abstraction level. The path is represented as a list of positions at the same abstraction level. A base-level path must be obstacle-free and hence is a route that the robot can follow. A path at a higher level of abstraction, on the other hand, is the result of an A* search that minimizes the length as well as the obstacleness of the result. It does not correspond to a particular list of base-level positions.

3.3 Plan execution

In order to follow an abstract path, the robot must use an obstacle avoidance procedure that may lengthen the route. As long as there exists a path that connects the start and goal positions, the obstacle avoidance procedure can bring the robot to its destination. Therefore, any abstract plan is completable and executable – even when blocked by obstacles. Obstacle avoidance is not a smart navigation method, but it can always substitute for missing details in an abstract plan. The quality of a plan $\mathcal{P}$ is defined as follows:

$$Quality(\mathcal{P}) = \frac{length(\mathcal{P}_{opt})}{length(\mathcal{P})}$$

where $length(\mathcal{P})$ is the length of the plan and $\mathcal{P}_{opt}$ is the optimal plan. Note that the higher the level of abstraction the lower the quality of the plan. At the same time, high-level abstract planning reduces (exponentially) the search space and hence it is performed much faster.

The notion of executable abstract plans – regardless of their arbitrary level of detail – is made possible by using plans as advice that direct the base level execution mechanism but does not impel a particular behavior. This idea was promoted by [Agre and Chapman, 1990] and was experimentally supported by [Gat, 1992]. In practice, uncertainty makes it impossible to use plans except as a guidance mechanism.

3.4 Performance with perfect vision

We now examine the performance of the abstract planner under the assumption of perfect domain description. Figure 4 shows the paths generated by the path finder when activated with the start and goal positions being the lower left and upper right corners respectively. The upper frame shows (by the large squares drawn in broken line) an abstract plan at level 3. The quality of the plan, 0.826, is determined by the length of the route the robot would have followed if guided by this plan (shown in the figure by a heavy broken line) compared to the length of the shortest route. The lower frame shows a more precise abstract plan with segments at levels 0 and 1. Notice that in this example the quality of the plan reached 0.985 – almost as good as the quality of the shortest path.

The typical performance of the planner is summarized by its performance profile in Figure 5. The graph shows the expected quality of the plan as a function of run-time. When run until completion, the expected quality of the plan produced by the abstract planner is 0.93. At the same time, its expected completion time is only 27% of the the expected run-time needed to compute the optimal path using the standard $\mathcal{A}^*$ algorithm. These figures show that anytime algorithms offer not only more flexibility but also a better cost/performance ratio.

3.5 Performance with imperfect vision

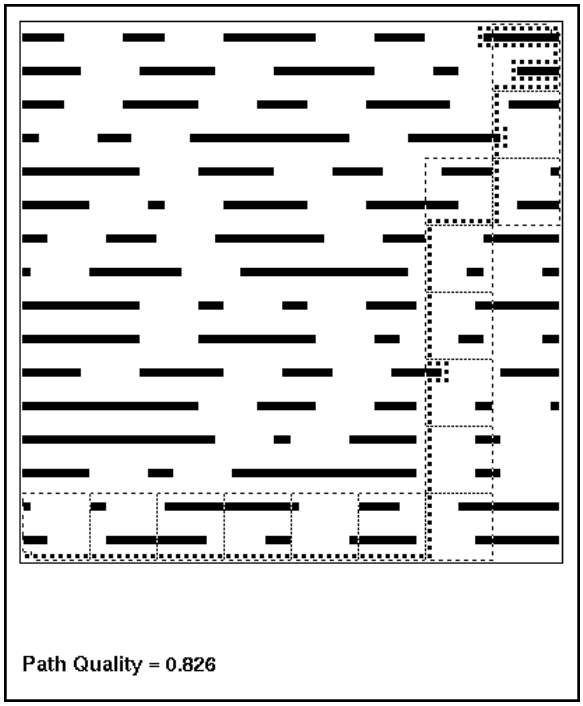
We now turn to examine the effect of vision errors on the quality of planning. The following demonstration is based on vision quality of 0.96. This figure is a measure of the sensor's noise level as described in the previous section. The physical domain is identical to the one used in the previous example, however, the map constructed by the vision module is erroneous.

Figure 6 shows snapshots of the plan generated by the algorithm and their qualities. Notice that as a result of lower quality of sensing, the quality of the initial plan is only 0.760 compared to 0.826 with perfect vision. On average, as a result of the error in the domain description, the planner produces plans of lower quality.

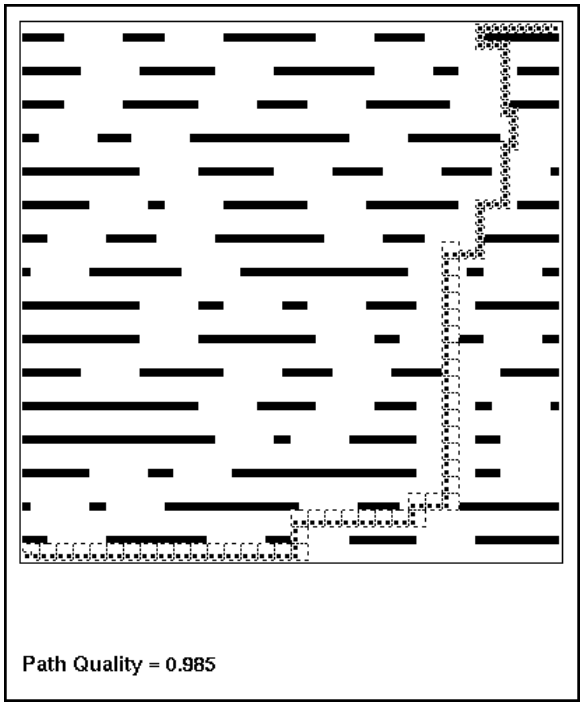
Based on statistics gathered by running the planning algorithm many times on randomly generated domains, we derived its conditional performance profile. It describes the expected quality of a plan based on the quality of the domain description and run-time. Figure 7 shows the conditional performance profile. Each curve shows the expected plan quality as a function of run-time for a particular quality of vision.

4 Compilation of sensing and planning

The compilation of anytime algorithms is a process that essentially extends the idea of functional composition to anytime computation. It allows the programmer to compose a system using anytime algorithms as components without dealing directly with the time allocation problem. To explain the compilation process we must first make a distinction between



(a) Level 3 plan



(b) Level 1/0 plan

Figure 4: Abstract plans with perfect vision

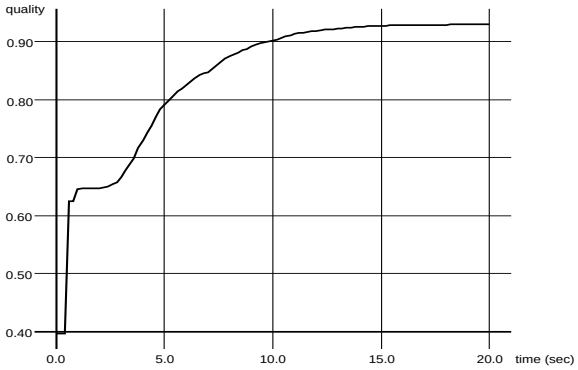


Figure 5: The performance profile of the planner

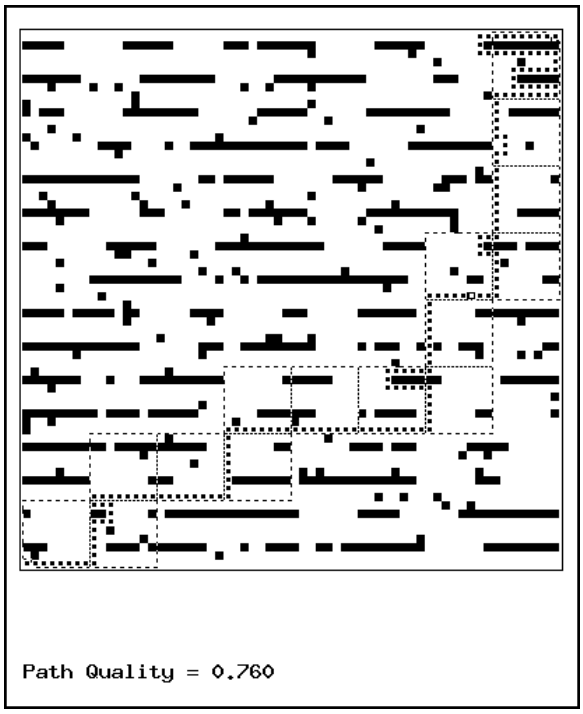
interruptible algorithms and *contract* algorithms. Interruptible algorithms produce results of the quality "advertised" by their performance profiles even when interrupted unexpectedly; whereas contract algorithms, although capable of producing results whose quality varies with time allocation, must be given a particular time allocation in advance. The greater freedom of design makes it easier to construct contract algorithms than interruptible ones. The compilation process creates a contract algorithm. In those cases where it is necessary to use an interruptible algorithm, the contract algorithm can be transformed into an interruptible one using a construction method presented in [Russell and Zilberstein, 1991]. Having made this distinction, we can define compilation as follows:

Definition: *Compilation of anytime algorithms is the process of deriving a contract algorithm with an optimal performance profile from a program composed of several anytime algorithms whose conditional performance profiles are given.*

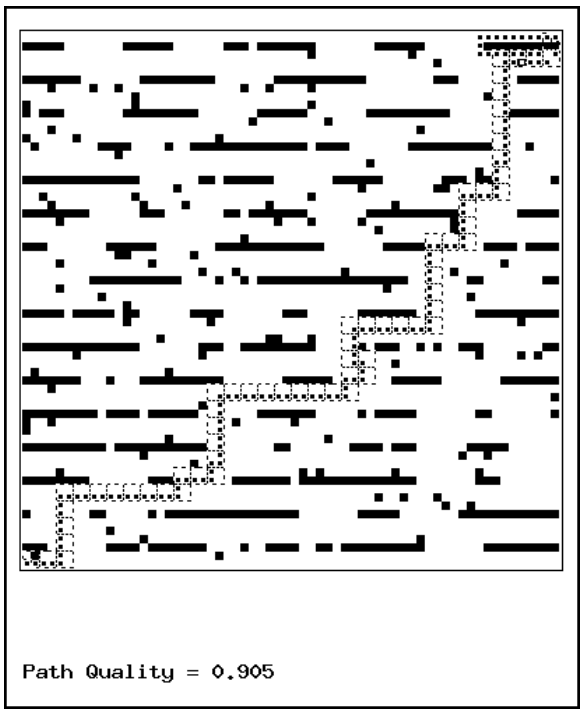
The input to the compiler includes a user defined function that we call the *program schema*. It looks like a regular code but some of the functions it uses may be anytime algorithms. The compiler also gets a set of *conditional performance profiles* stored in a library. The task of the compiler is to produce a new version of the program that includes code to control the distribution of time between the components so as to maximize the overall performance for any given time allocation. It also creates a performance profile for the complete system, based on the optimal time allocation.

The hardest part of the compilation is finding the time allocation to the components that yields maximal quality. This problem has been solved in [Zilberstein, 1993] with respect to a rich compositional language. In the case of composition of n modules, the global optimization problem is shown to be NP-complete by transformation from the partially ordered knapsack problem (which is known to be NP-complete in the strong sense [Garey and Johnson, 1979]). However, a local compilation technique, that works on a single program structure at a time, significantly improves the efficiency of compilation and is proved to yield optimal performance for a large set of program structures. The composition of planning and sensing is a simple example of such program. It is represented by the following program segment:

(find-path Start Goal (get-domain-description Sensor))



(a) Level 3 plan



(b) Level 1 plan

Figure 6: Abstract plans with imperfect vision

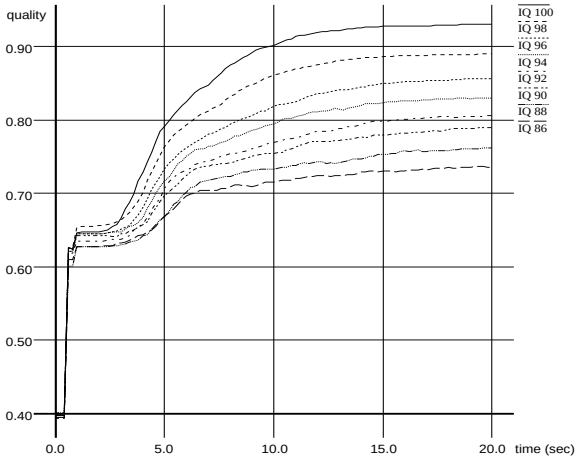


Figure 7: The conditional performance profile of the planner

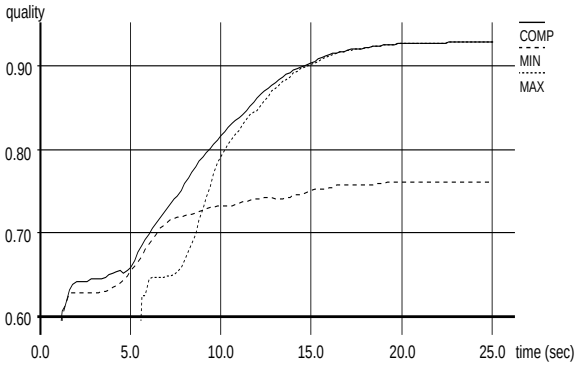


Figure 8: Compilation of vision and planning

Figure 8 shows the performance profile that we got by compiling this program. Also shown in that figure are the performance profiles of two other modules: MIN, that allocates to vision a minimal amount of time, T_a, and MAX, that allocates to vision a maximal amount of time, T_b. The compiled performance profile is superior to both. The reader can find a more detailed description of the compilation process in [Zilberstein, 1993].

5 The run-time system

Systems composed of anytime algorithms require constant monitoring. The compilation process provides the necessary meta-level information to make the run-time monitoring more efficient. In this section we explain how the run-time system controls the time allocation to the anytime modules.

The optimization of the long-term behavior of the robot is performed by dividing a complex task into a series of small sensing, planning and plan execution episodes called *frames*. For each frame, we use the anytime sensing and planning modules described earlier. Since, in many cases, sensing capability is limited to a small, local segment of the environment, it is only natural to break the navigation problem into such frames.

The task of the meta-level control is to determine the optimal initial contract time for each frame. This decision – *inter-frame optimization* – is made in the following way: let t be the current time (real-time since the beginning of the execution of the task), let f_t be the (estimated) number of frames left at time t for planning and execution, let t_c be the contract time for the next cycle of planning and execution, and let e_t be the energy used so far for plan execution. Then,

$$t_c = argmax_{t_i}\{VOT(t, f_t, t_i) - COE(e_t, f_t, t_i)\}$$

where VOT is the expected value of the task and COE is the expected cost of energy. Note that both the performance profile of the system and a model of the environment are necessary in order to compute these functions.

Once an initial contract time is determined, the system starts allocating resources to sensing, planning and plan execution. At the same time it continues to monitor the performance of the anytime modules. This constant monitoring is necessary because of the uncertainty concerning the *actual* quality of plans and the *actual* time necessary to execute them. The purpose of the meta-level control in this phase is to reach an optimal plan quality for the next frame while executing a previously derived plan. For this purpose, it can modify the initial contract time. This decision – *intra-frame optimization* – is made in the following way: the monitor determines at each point whether planning is ahead of or behind expectations by comparing the (estimated) plan quality to the quality advertised by the performance profile. It also determines whether plan execution is ahead of or behind expectations by comparing the (estimated) execution time to the frame contract time. If planning is ahead of expectations and plan execution is behind, the monitor accelerates plan execution by allocation more resources (energy) to plan execution. If planning is behind and plan execution is ahead, the monitor slows down plan execution by reducing resource consumption.

This monitoring strategy can be modified in various ways. For example, one can consider planning more than one frame ahead, when plan execution is slow. Another possibility in this case is to replan part of the plan that is being executed to accelerate plan execution. However, our experiments with the above domain show that the monitoring strategy that was implemented is sufficient in order to achieve (within 4% error) the optimal task value that the system computes when presented with the task.

6 Conclusion

We have presented a method to construct robotic systems and to optimize their performance. The method is based on developing the performance components of the system as anytime algorithms. The control of the anytime components of the system is efficiently implemented using off-line compilation and run-time monitoring. An implementation of the model was presented that solves a particular path planning problem. Our approach offers several improvements over traditional *ad hoc* techniques used to construct robotic systems: it is an optimizing rather than satisficing method; it allows complex planning to be used in real-time robotic systems; it helps construct systems when resource availability is unknown at design time; and it efficiently integrates sensing, planning and plan execution.

The anytime abstract planning algorithm that was presented produced high quality results with time allocation that was much shorter than the total run-time of a standard search algorithm. This shows that the flexibility of anytime algorithms does not necessarily require a compromise in overall performance, even with a fixed time allocation.

By further generalizing the various components of the system, we aim at constructing a general, flexible mechanism for developing self-optimizing autonomous robots whose perception, decision making and action are implemented as anytime modules. This approach offers a more realistic theoretical foundation for robot planning by addressing the problems of uncertainty, limited computational power, and imprecise sensing.

Acknowledgements

Support for this work was provided in part by the National Science Foundation under grants IRI-8903146 and IRI-9058427 (Presidential Young Investigator Award).

References

[Agre and Chapman, 1990] P. E. Agre and D. Chapman. What Are Plans for? In *Robotics and Autonomous Systems*, 6:17–34, 1990.

[Dean and Boddy, 1988] T. Dean and M. Boddy. An Analysis of Time-Dependent Planning. In *Proc. Seventh National Conference on Artificial Intelligence*, pp. 49–54, Minneapolis, Minnesota, 1988.

[Garey and Johnson, 1979] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, California: W. H. Freeman and Company, 1979.

[Gat, 1992] E. Gat. Integration Planning and Reacting in a Heterogeneous Asynchronous Architecture for Controlling Real-World Mobile Robots. In *Proc. Tenth National Conference on Artificial Intelligence*, pp. 809–815, San Jose, California, 1992.

[Horvitz, 1987] E. J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Proceedings of the 1987 Workshop on Uncertainty in Artificial Intelligence*, Seattle, Washington, 1987.

[Lozano-Pérez and Brooks, 1984] T. Lozano-Pérez and R. A. Brooks. in *Solid Modeling by Computers*, M. S. Pickett and J. W. Boyse, Eds. pp. 293-327, Plenum, New York, 1984.

[Russell and Zilberstein, 1991] S. J. Russell and S. Zilberstein. Composing Real-Time Systems. In *Proc. 12th International Joint Conference on Artificial Intelligence*, pp. 212–217, Sydney, Australia, 1991.

[Zilberstein and Russell, 1992] S. Zilberstein and S. J. Russell. Efficient Resource-Bounded Reasoning in AT-RALPH. In *Proc. First International Conference on Artificial Intelligence Planning Systems*, pp. 260–266, College Park, Maryland, 1992.

[Zilberstein, 1993] S. Zilberstein. *Operational Rationality Through Compilation of Anytime Algorithms*. Ph.D Dissertation, Computer Science Division, University of California, Berkeley, 1993.