Control of Mobile Robots Using Anytime Computation

Shlomo Zilberstein and Stuart J. Russell
Computer Science Division
University of California
Berkeley, California 94720

Abstract

Anytime algorithms are algorithms whose quality of results improves gradually as computation time increases. They have attracted growing attention in recent years as a key mechanism for time-critical planning and control of autonomous robots. By introducing computation time as a degree of freedom, anytime algorithms define a scheduling problem involving the activation and interruption of the anytime components. We have implemented a prototype of AT-RALPH in which an off-line compilation process together with 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*. These profiles characterize the performance of each elementary anytime algorithm as a function of run-time and input quality. We have also extended the notion of gradual improvement to sensing and plan execution. While control of sensing is very similar to control of anytime computation, the timing of plan execution can be controlled by varying the resources (such as energy) consumed by the robot. The result is an efficient, flexible control for autonomous robots that optimally exploits the tradeoff between time and quality in planning, sensing and plan execution.

Introduction

Anytime algorithms introduce a tradeoff between computation time and quality of results. This degree of freedom is especially useful when developing control mechanisms for autonomous mobile robots. The flexibility offered by anytime algorithms allows accurate sensing and complete planning when time is available and coarse fast sensing and planning under time pressure. However, as time allocation becomes a degree of freedom, intelligent incremental scheduling and constant monitoring are necessary in order to guarantee the optimal operation of the robot.

We have implemented a prototype of AT-RALPH (Anytime Rational Agent with Limited Performance Hardware) in which the scheduling problem is solved through an off-line compilation process together with a run-time monitoring component. In order to reason efficiently about time allocation, we introduce *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 on run-time only [Dean and Boddy, 1988].

The rest of this paper explains our approach in detail and demonstrates its implementation. We start with a definition of a path planning problem. Then we describe the two components of the robot that solve the problem: anytime planning and anytime sensing. The next section explains the compilation scheme that optimally integrates the two components. Finally we present the run-time system and summarize the benefits of our approach.

The path planning problem

In order to demonstrate our compilation scheme and meta-level control, we have used one of the fundamental problems facing any autonomous mobile robot: the capability to plan its own motion. This section describes the path planning problem and the simulated environment in which the robot is situated.

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 such a map for a small section of the environment. The accuracy of the domain description, produced by the vision mechanism, depends on the time allocated to the vision module. The environment[1] is composed of a matrix of elementary positions. The robot can move between adjacent cells of the matrix at a varying speed. The speed affects the execution time of a 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

[1] Similar to the RALPH world [Parr, 1992].

a particular position and perform a certain job. Associated with each task is a value function that determines the value of completing 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.

Anytime planning

The path planning algorithm that we use is an extension of A^*. 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 at a lower level of abstraction.

Abstract description of the domain Each level of abstraction, n, 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. It is simply the proportion of the matrix that is covered by obstacles.

A general position in this two dimensional domain has therefore three components: $(x \ y \ l)$: where x and y are the coordinates and l is the level of abstraction. The position (3 3 1), for example, corresponds to the following set of base level positions: (6 6 0) (6 7 0) (7 6 0) (7 7 0). If one of these positions contains an obstacle and the rest are free, then the obstacleness of (3 3 1) is 0.25.

The general path finder The *general path finder* (gpf) is a search procedure that returns the best path between any two positions (at any abstraction level). The path is represented as a list of positions at the abstraction level of the start and goal positions. A base-level path is never blocked by obstacles and hence is an exact route that the robot can follow. A path at a higher level of abstraction however is the result of a best-first search that minimizes the length as well as the obstacleness of the result but does not correspond to a particular list of base-level positions that the robot can follow. In order to follow an abstract path, the robot must use an obstacle avoidance procedure that may lengthen the route. Since the obstacle avoidance procedure can always bring the robot to its destination, any abstract plan is completable and executable – even when blocked by obstacles. Clearly, obstacle avoidance is not a smart navigation method but it can always substitute missing details in abstract plans. Therefore, the higher the level of abstraction the lower the quality of the plan. At the same time, high-level abstract planning reduces (exponentially) the distance to the goal and hence it is performed much faster.

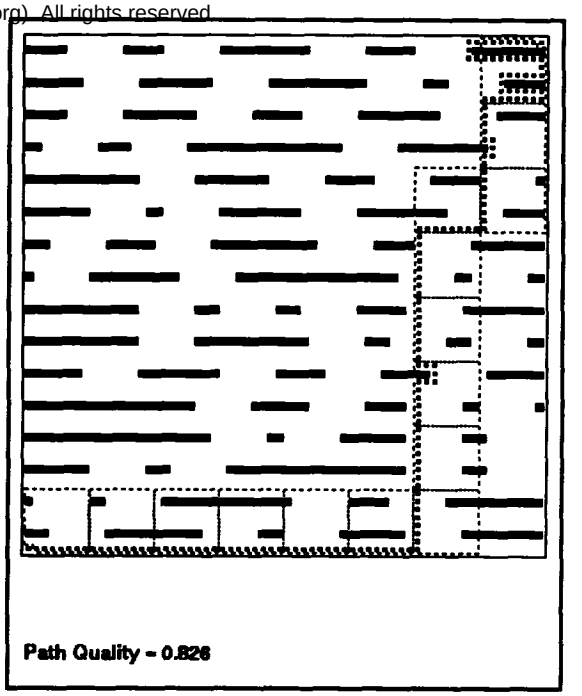

Figure 1: High-level abstract plan

Anytime abstract planning The interruptible *anytime planner* (atp) constructs a series of plans whose quality improves over time. It starts with a plan generated by the general path finder at the highest level of abstraction. Then, it repeatedly refines the plan created so far by picking the *worst segment* of the plan, dividing it into two segments (of identical length), and replacing each one of 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 nice property of the algorithm is that the length of each segment in the intermediate plan is invariant. It depends only on the length of the initial path at the highest level of abstraction. As a result, the run-time of the refinement step is (approximately) the same for any segment of the plan regardless of the level of abstraction.

Demonstration

Below is a sample run of the planning algorithm in a domain with rectangular obstacles. We start with perfect input quality (that is, assuming no vision errors).

Figure 1 shows the path generated by the general path finder when activated at the highest abstraction level (3 in this example) with the start and goal posi-

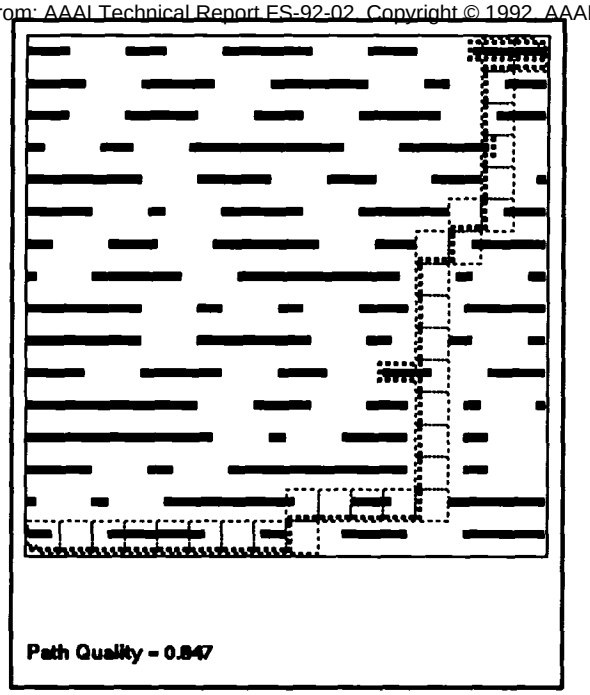

Figure 2: Plan at level 2

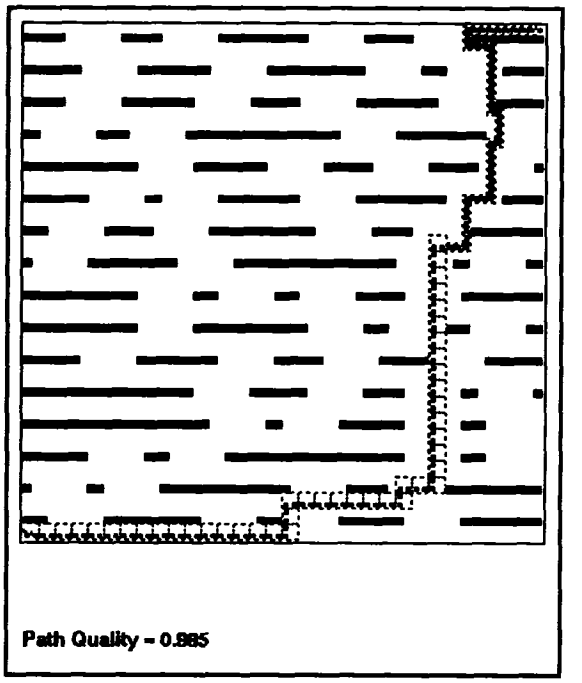

Figure 3: Plan at levels 0 & 1

tions being the lower left and upper right corners respectively. The abstract plan is shown by a broken line. The quality of the plan (0.826) is determined by the length of the route the robot follows when guided by this plan (shown in the figure by the small, filled squares) compared to the length of the shortest route.

Figures 2 and 3 show snapshots of the abstract plans generated by the algorithm and their qualities. Notice that the quality of the plan improves monotonically reaching 0.985, almost the quality of the shortest path. In addition to being an interruptible anytime algorithm and to reaching high quality of results, the total run-time of the abstract planner is much shorter than the time needed to compute the optimal (shortest) path using the standard A^* algorithm.

Figure 4 shows the performance profile of the abstract planner when fed with accurate domain description. It gives the expected quality of the plan as a function of run-time. We now turn to an example where the quality of the domain description is 0.96. As we indicated earlier, the domain description is produced by the vision module. The exact meaning of vision quality is defined in the next section. For the purpose of understanding the example, it is enough to think about it as a measure of the sensor's noise level. The physical domain is identical to the one used in the previous example.

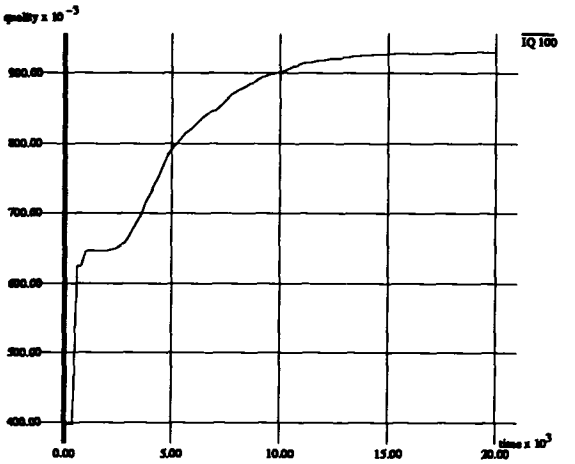

Figure 4: Performance profile of atp

Figure 5 shows the initial path generated by the general path finder. Notice that the quality of the plan, as a result of lower quality of sensing, is only 0.760 compared to 0.826 with perfect vision.

Figures 6 and 7 show snapshots of the plans gener-

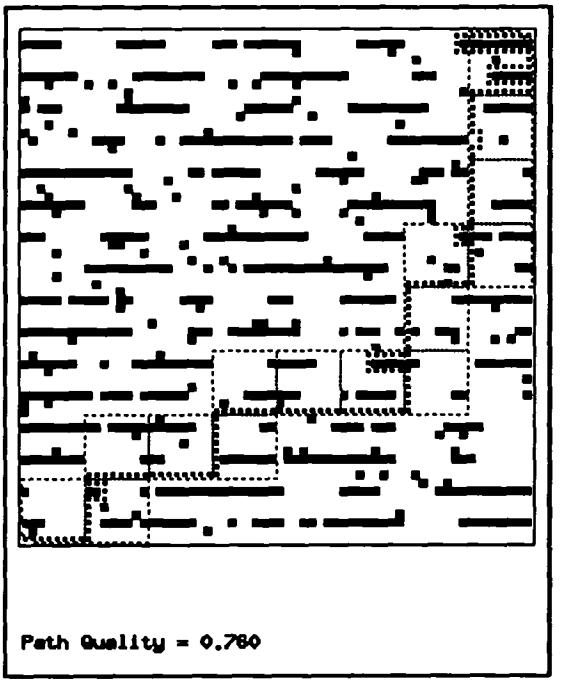

Figure 5: Abstract plan with vision errors

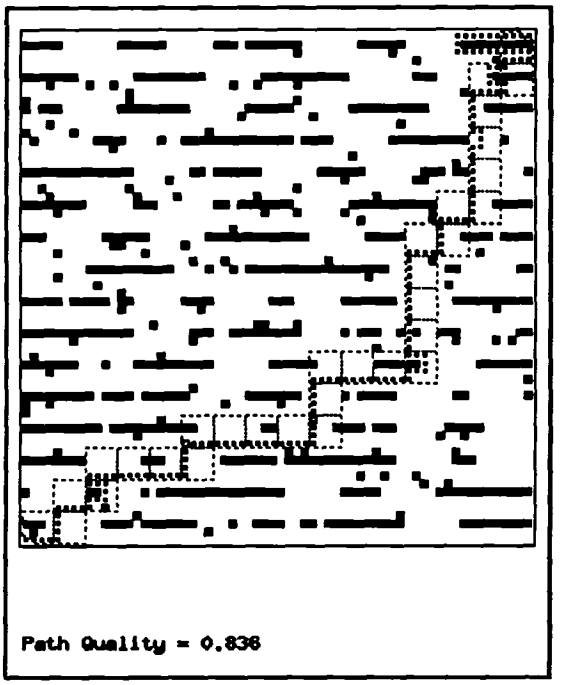

Figure 6: Plan at level 2

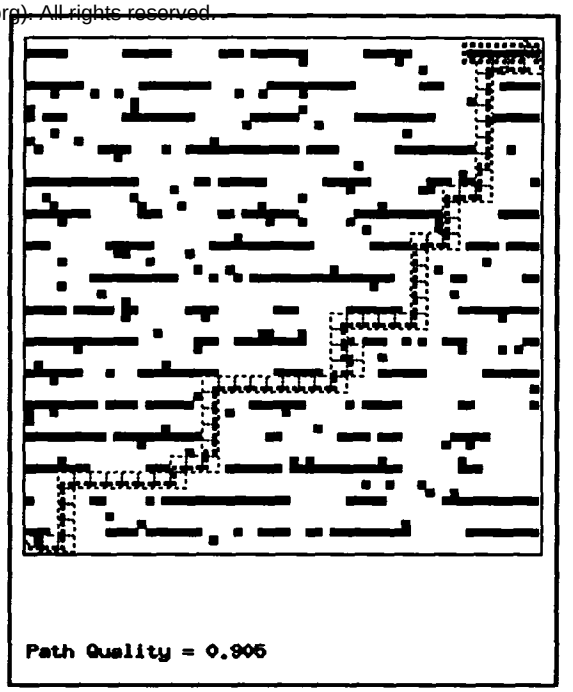

Figure 7: Plan at level 1

ated by the algorithm and their qualities. In this particular example, the planner eventually reached a plan of the same quality as in the perfect vision case, but it took more time. In many cases, however, as a result of the error in the domain description, the planner terminates with a plan of lower quality. Based on statistics gathered by running the planning algorithm many times with randomly generated domains, we derived its conditional performance profile. It describes the expected quality of a plan based on domain description quality and run-time. The conditional performance profile and its use are discussed later in the paper.

Anytime sensing

One of the goals of this work has been to extend the notion of gradual improvement to sensing. The assumption that sensors produce a perfect domain description, as much as the assumption of perfect planning, is not only inefficient but normally impossible for an autonomous robot operating in a real-world domain. Therefore, the qualitative analysis of the effect of sensory noise on the quality and performance of the rest of the system is essential in the construction of robots. This section describes the model of anytime vision that we implemented. It produces a domain description whose quality expresses 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.

Figure 8 shows the performance profile of the simulated vision module. It is characterized by several pa-

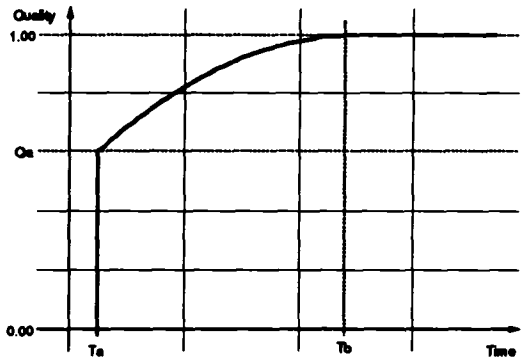

Figure 8: Performance profile of the vision module

rameters. T_a is the minimal amount of time needed for the sensor to produce an initial domain description with quality Q_a. For 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 1.00. We model this improvement using a hyperbola with maximum at $(T_b, 1)$.

Notice that when a sensor is interrupted at any time shorter than the minimal time needed to produce useful data, it is still possible for the system to operate using prior knowledge on the domain. For example, one can assume that every position is free in a domain with scarce obstacles. Obviously, this assumption is only allowed when the robot has a control mechanism the would prevent any damage when the assumption is wrong.

Compilation of sensing and planning

The compilation of anytime algorithms, a central concept in our system, is an automated 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 *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 simple construction method presented in [Russell and Zilberstein, 1991].

We now turn to a detailed look at the compilation process, starting with its definition:

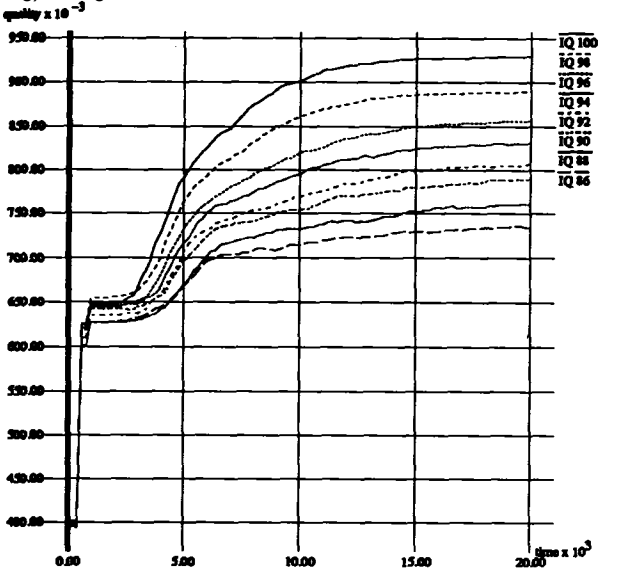

Figure 9: Conditional performance profile of atp

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 performance profiles are given.*

The input of the compiler includes a user defined Lisp function that we call *program schema*. It may look like a regular Lisp code but some of the functions it calls may be anytime algorithms (the user however may ignore the time allocation problem). 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 optimal time allocation.

Conditional performance profiles To be able to properly combine anytime algorithms one has to take into account the fact that the quality of the results depends not only on time allocation but also on properties of the input, most notably its quality. Performance profiles in our system are therefore conditional. They consist of mappings from input quality and run-time to probability distribution of output quality:

$$\mathcal{CPP} : Q_{in} \times T \rightarrow Pr(Q_{out})$$

Figure 9 shows the conditional performance profile of the abstract planner. Each curve shows the expected plan quality as a function of run-time for a particular quality of vision.

Compilation In the current version of AT-RALPH we have solved the compilation problem for the case

where compound (non-elementary) anytime modules are restricted to *composition* of other anytime algorithms. An equivalent assumption is that the code of a compound algorithm can be written as a *straight line program* with anytime algorithms as basic operations. A straight line program is a sequence of expressions of the form *(setf u (f $v_1 ... v_n$))* where u is a new variable and $v_1 ... v_n$ are program arguments or existing variables. There is a trivial one-to-one mapping between functional composition and straight line code. The composition of planning and sensing is a simple example of straight line code.

The main component of the compiler is a hill-climbing time allocation algorithm. It starts with an equal amount of time allocated to each anytime module. Then it considers trading s time units between two modules so as to increase the expected quality of the results. As long as it can improve the expected quality, it trades s time units between the two modules that have maximal effect on output quality. When no such improvement is possible with the current value of s, it divides s by 2 until s reaches a certain minimal value, ϵ. At that point, it reaches a local maxima and returns the best time allocation it found. As with any hill-climbing algorithm, it suffers from the problem of converging on a local maxima.

Time Allocation Algorithm

```
for each Q_in ∈ QUALS-TABLE
    for each T ∈ TIME-TABLE
        s ← initial-resolution(T)
        t_i ← T/n   ∀i : 1 ≤ i ≤ n
        repeat
            while ∃i, j such that
                E(Q_out(Q_in, t_1, ..., t_i − s, ..., t_j + s, ..., t_n))
                  > E(Q_out(Q_in, t_1, ..., t_n))
                let i, j be the ones that maximize E(Q_out)
                t_i ← t_i − s
                t_j ← t_j + s
            end
            s ← s/2
        until s < ε
        CPP[Q_in, T] ← [Pr(Q_out), t_1, ..., t_n]
    end
end
```

Complexity The current version of the system uses discrete performance profiles stored in a two dimensional table (with input quality and run-time as indices). Linear interpolation is used to compute the performance distribution for points that do not match exactly one of the table entries. The time allocation algorithm therefore has to fill-in this two dimensional table. For each entry in the table, the complexity of the algorithm is $O(n^2 log(T/\epsilon))$, where n is the number of modules, T is the total run-time and ϵ is the lowest number of time-units traded between modules. Since the number of modules used to define a new function is normally small, the overall complexity of the algorithm is dominated by the accuracy of the compiled performance profile (a system parameter that determines the size of the table representing the performance profile).

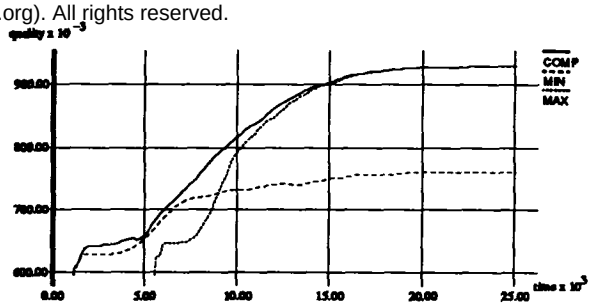

Figure 10: Compilation of vision and planning

Figure 10 shows the performance profile that we got by compiling sensing and planning. 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 algorithm is obviously superior to both.

In addition to calculating the performance profile of the system, the compiler inserts in the original function the necessary code to control internal time allocation. For more detail on the compilation process see [Zilberstein and Russell, 1992a].

The run-time system

This section explains how the run-time system controls the time allocation to the anytime modules. Figure 11 shows the data flow between the main components. 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 contract time. The other factors are the performance profile of the system, the model of the environment, and the quality of the current best plan.

The optimization of the behavior of the robot is performed by dividing the complete task into a series of small sensing, planning and plan execution episodes called *frames*. The meta-level control has 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 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 used by these functions.

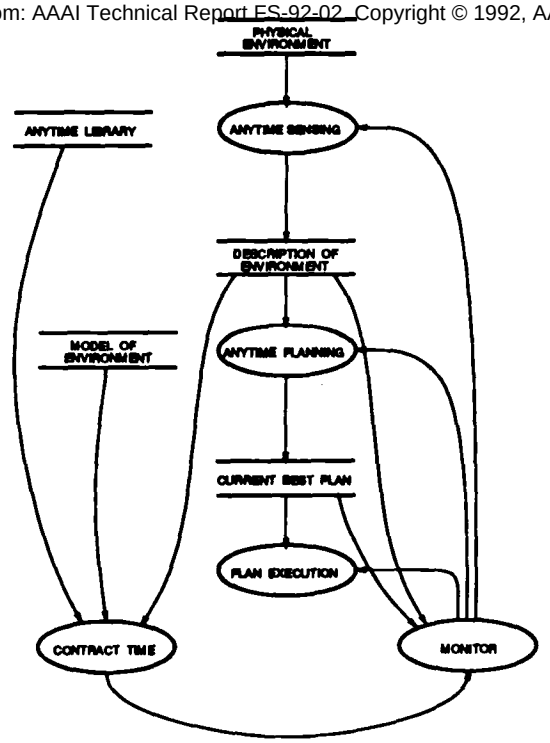

Figure 11: The run-time system

Once the initial contract time is determined, the system can start 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. Obviously, it can modify the initial contract time if necessary. The decision (intra-frame optimization) is made in the following way:

The monitor determines at each point whether planning is ahead of or behind schedule by comparing the (estimated) plan quality to the expected quality based on the performance profile. It also determines whether plan execution is ahead or behind. Figure 12 shows how the final decision is made. In this figure, + represents a process being ahead of schedule and − represents a process being behind schedule. If planning is ahead and plan execution is behind, the monitor accelerates plan execution by allocation more resources (energy) to plan execution. If planning is ahead and plan execution is behind, the monitor slows down plan execution by reducing resource consumption.

Figures 13 and 14 show the display of the run-time system. The first shows the planning frame that includes the best plan so far, its expected quality, the contract time, frame time, sensing time and its quality. The second shows the plan execution frame that includes the path followed by the robot, the current time and the energy consumed so far. It also shows the expected task completion time and value.

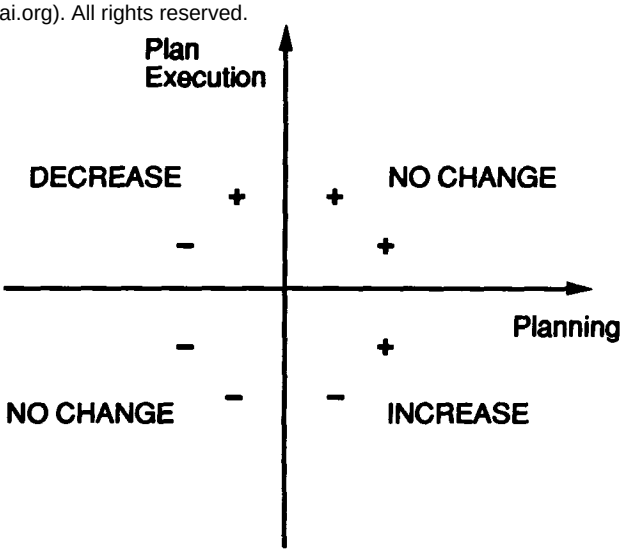

Figure 12: Intra-frame optimization

Conclusion

We have presented a method to optimize the performance of autonomous mobile robots based on compilation of anytime algorithms. The application of the method to a particular path planning problem produced encouraging positive results. The implementation also suggests a rather general methodology for developing interruptible anytime algorithms by first solving the problem at a high level of abstraction and then repeatedly refining the solution. Our approach offers several improvements over traditional *ad hoc* techniques used to construct autonomous robots: it is an optimizing rather than satisficing method; it helps construct real-time systems when resource availability is unknown at design time; it efficiently integrates sensing, planning and plan execution; and it provides machine independent real-time modules.

The anytime planning algorithm produce high quality results with time allocation that is much shorter than the total run-time of a standard search algorithm. This fact shows that the flexibility of anytime algorithms does not require a compromise in overall performance.

By further generalizing the various components of AT-RALPH 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.

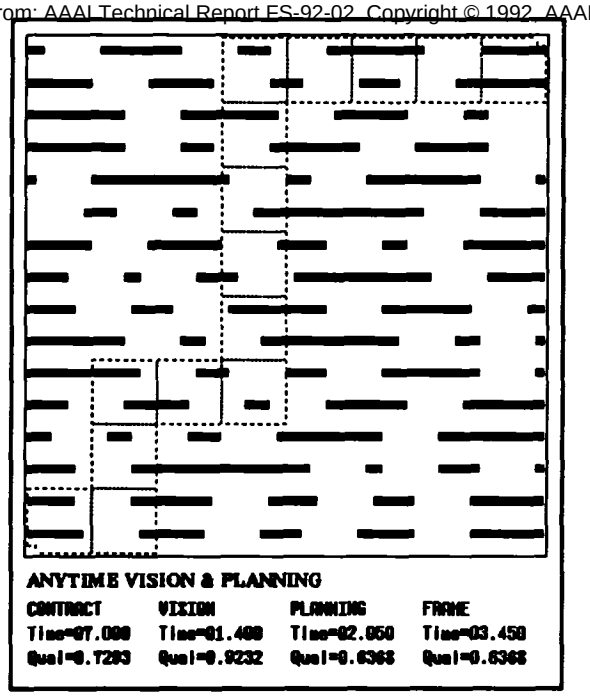

Figure 13: Planning and sensing frame

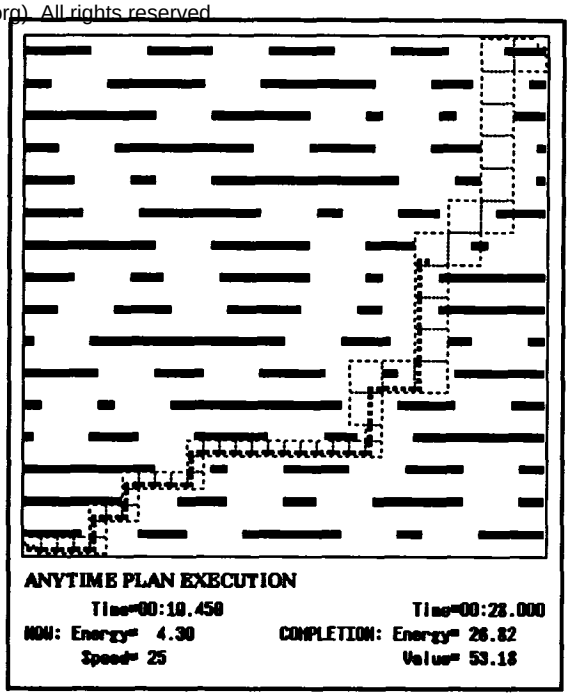

Figure 14: Plan execution frame

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). The first author was also supported by the Malcolm R. Stacey Scholarship.

References

[Boddy and Dean, 1989] M. Boddy and T. Dean. Solving time-dependent planning problems. Technical Report CS-89-03, Department of Computer Science, Brown University, Providence, 1989.

[Dean and Boddy, 1988] Thomas Dean and Mark Boddy. An Analysis of Time-Dependent Planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–54, Minneapolis, Minnesota, 1988.

[Hendler, 1989] James A. Hendler. Real Time Planning. In *Proceedings of the AAAI Spring Symposium on Planning and Search*, Stanford, California, 1989.

[Latombe, 1991] Jean-Claude Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, Norwell, Massachusetts, 1991.

[Lawler *et al.*, 1987] E. L. Lawler *et al.*, (Eds). *The traveling salesman problem: a guided tour of combinatorial optimization.* Wiley, New York, 1987.

[Parr *et al.*, 1992] Ronald Parr, Stuart J. Russell and Mike Malone. *The RALPH System.* UCB Technical Report (to appear).

[Russell and Wefald, 1991] Stuart J. Russell and Eric H. Wefald. *Do the Right Thing: Studies in limited rationality.* MIT Press, Cambridge, Massachusetts, 1991.

[Russell and Zilberstein, 1991] Stuart J. Russell and Shlomo Zilberstein. Composing Real-Time Systems. In *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, 1991.

[Zilberstein, 1991] Shlomo Zilberstein. Integrating Hybrid Reasoners through Compilation of Anytime Algorithms. In *Proceedings of the AAAI Fall Symposium on Principles of Hybrid Reasoning*, pages 143-147, Pacific Grove, California, November 1991.

[Zilberstein and Russell, 1992a] Shlomo Zilberstein and Stuart J. Russell. Efficient Resource-Bounded Reasoning in AT-RALPH. To appear in *Proceedings of the First International Conference on Artificial Intelligence Planning Systems*, College Park, Maryland, June 1992.

[Zilberstein and Russell, 1992b] Shlomo Zilberstein and Stuart J. Russell. Reasoning About Optimal Time Allocation Using Conditional Performance Profiles. To appear in *Proceedings on the AAAI Workshop on Implementation of Temporal Reasoning*, San Jose, California, July 1992.