In the 2000 AAAI SSS 'Real-Time Autonomous Systems'

RT-MLab: Really Real-Time Robotics

Karen Zita Haigh, David J. Musliner, Sunondo Ghosh
Honeywell Technology Center
3660 Technology Drive
Minneapolis, MN 55418
{khaigh,musliner,sghosh}@htc.honeywell.com

Abstract

A common misconception about real-time computing is that it is equivalent to "fast computing." Many robotics examples in recent history have shown that without rigorous understanding of the resource requirements and interactions of the software, these systems are doomed to failure.

In this paper, we discuss our application of rigorous real-time analysis and execution methods to Georgia Tech's MissionLab system for robot tasking and control [Endo *et al.* 1999; MacKenzie, Arkin, & Cameron 1997]. In this paper, we describe RT-MLab, which combines MissionLab with Honeywell's MetaH real-time analysis and execution tool to provide real-time schedulability analysis and reliable real-time execution support for robot behaviour configurations.

Introduction

For decades, robotics researchers have pushed the boundaries of computer science and computing hardware, striving to build intelligent control systems that are smart enough and fast enough to control robots moving through complex, dynamic, real-world environments. One of the key constraints on these systems is the need to operate in real-world time, rather than simulated or virtual time: the robot's control system must detect and react appropriately to a continuously changing world that is not entirely controllable, and cannot be slowed down. The progression of Moore's Law and the advent of megaflop processors has not changed the fundamental situation: we still need robot control algorithms and architectures that behave in timely ways. That is, we need real-time robotics.

A common misconception about real-time computing is that it is equivalent to "fast computing" [Stankovic 1988]. As a result, most robotics researchers to date have treated the timing behaviour of their systems in an *ad hoc* fashion, re-engineering algorithms when testing indicates that they are running too slowly or causing errors. Without a rigorous understanding of the resource requirements and interactions of their software, these testing-based efforts are doomed to the same sort of failures we've seen in many other well-tested embedded systems (e.g., Space Shuttle launches delayed by synchronization errors, Mars Rover and Deep Space One failures caused by improperly-controlled multitasking).

In real-time computing, the correctness of the system depends not only on the logical result of the computation, but also on the time at which the results are produced [Stankovic 1988]. The real-time systems community has studied these issues and developed techniques to address them using formal, rigorous techniques [Shin & Ramanathan 1994; Krishna & Shin 1997]. In this paper, we discuss our application of rigorous real-time analysis and execution methods to Georgia Tech's MissionLab system for robot tasking and control [Endo *et al.* 1999; MacKenzie, Arkin, & Cameron 1997].

MissionLab allows a user to easily construct a set of robot behaviours, download the behaviours to the robot, and execute the behaviours while observing feedback data. Prior to our efforts, the user could construct any set of behaviours, combining as many complex computations as he wished, and each behaviour would simply run "as fast as it could." Clearly, it is possible to assemble behaviours that overwhelm the robot's computing system and will not meet the real-time constraints imposed by the environment. For example, if the robot's computer cannot run the obstacle detection behaviour quickly enough, the robot will run into objects.

Our work with MissionLab has three primary objectives:

1. To develop automatic real-time analysis methods that assess whether a user-defined configuration of robot behaviours will be executed in a timely fashion on a particular robot.
2. To provide a real-time execution environment that

supports predictable, timely execution of behaviour configurations.

3. To demonstrate the general applicability of hard real-time performance guarantees for robots.

The system we developed, RT-MLab, combines MissionLab with Honeywell's MetaH real-time analysis and execution tool [Binns & Vestal 1993; Vestal 1997; 1998]. The MetaH toolset combines modelling, performance analysis, reliability, and partition security with automatic tailoring of efficient middleware services for embedded computer systems.

RT-MLab looks almost exactly like the original MissionLab system from the user's perspective. Via the original GUI, the user constructs a configuration of behaviours. RT-MLab analyzes the behaviours to determine whether they meet the timing constraints imposed by the hardware and execution environment. If the behaviours are feasible, they are compiled with MetaH glue code and downloaded to the MetaH execution environment, providing reliable real-time execution that is guaranteed to meet the specified timing requirements. The result: *a robotic tasking and control system that guarantees it will correctly execute any behaviour configuration it allows the user to generate.*

Architecture

MissionLab

MissionLab is an end-to-end robot behaviour specification and execution toolset. Figure 1 shows the architecture of the overall MissionLab system.

The human operator specifies a mission by drawing a finite-state diagram using the *Configuration Editor*. Figure 2 shows a behaviour configuration for a can-collecting robot. The end user selects behaviours from a library created by a system designer.

When the behaviour configuration is ready, the user binds it to a robot type. At this point, particular behaviours are bound to particular hardware capabilities. For example, a "detect human" behaviour might be bound to an infra-red sensor on one robot, and to a camera-based face-detection algorithm on another robot.

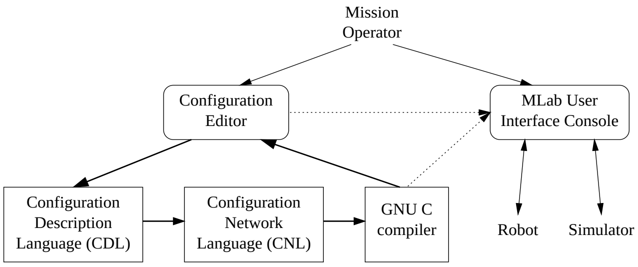

Figure 1: MissionLab Architecture.

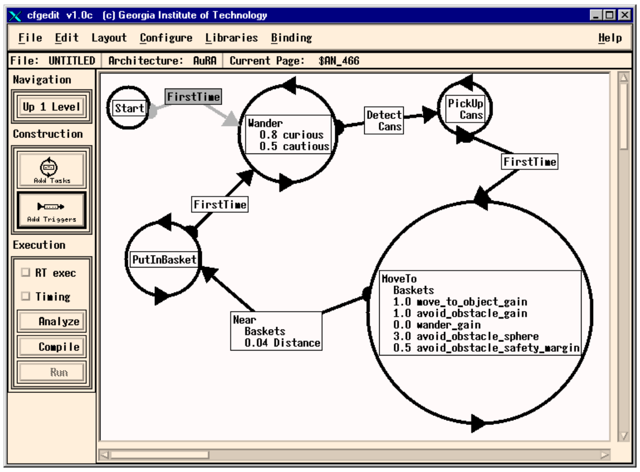

Figure 2: The Configuration Editor.

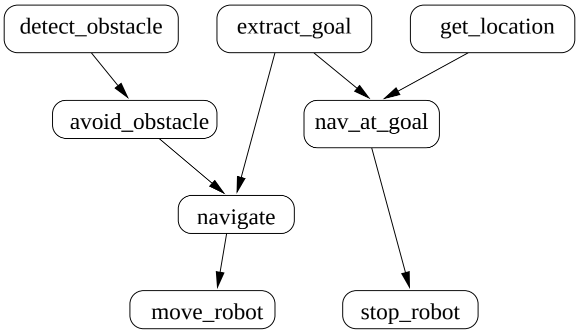

Figure 3: A simplified network of CNL nodes for a "(goto x,y)" behaviour configuration.

When the user clicks the "compile" button, MissionLab uses a series of compilers to generate a binary executable. MissionLab first compiles the Configuration Description Language (CDL) representation of the robot behaviours into the *Configuration Network Language (CNL)*. Each behaviour shown to the user is bound to several CNL nodes. For example, the user-level behaviour "(goto x,y)" requires the CNL nodes for obstacle detection, obstacle avoidance, localization, goal identification, and other functions. Figure 3 shows a very simplified CNL network for the "(goto x,y)" behaviour. The arrows between CNL nodes refer to the flow of information in the network. After compiling the user's behaviour configuration into CNL, MissionLab then compiles the CNL code into C, and then the C code into a target binary.

The compilation procedure generates two "logical units" of information, as shown in Figure 4. The first is the *behaviour*, which forms the basis of the end user's configuration. The second, generated by the CDL-to-CNL compiler, is a CNL *node*. Each behaviour maps to several nodes, and each node may be required by several behaviours. Each node runs as a light-weight thread in

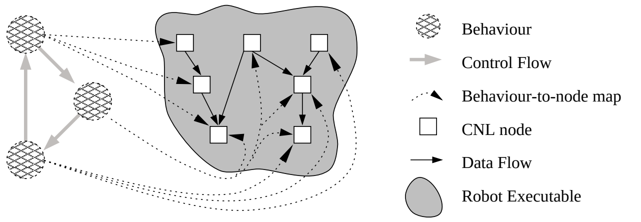

Figure 4: Behaviours and Nodes in MissionLab. Each node runs as a light-weight thread in the executable.

the executable. At execution time, each thread (node) in the CNL network runs continuously, generating a new output value for each set of new input values.

The final step is to run the executable. The user clicks the "run" button and chooses to execute on either the MLab simulator or a particular real robot. The MLab GUI, shown in Figure 5, allows the user to monitor the execution.

RT-MLab

RT-MLab is a rebuilt version of MissionLab with new capabilities including:

- Automatically calibrating CNL node runtimes.
- Analyzing configurations of behaviours for real-time execution feasibility.
- Predictably executing behaviour configurations and enforcing guaranteed real-time timing constraints.

From the user's perspective, the interaction with the robots is the same, with one added functionality: before compiling the behaviour configuration into a binary, the user clicks "analyze" in the configuration editor, and MetaH will analyze the configuration for feasibility. That is, it checks to see if all of the process timing constraints and communication interconnects form a set of processes that can be successfully executed on the robot, without violating any timing constraints.

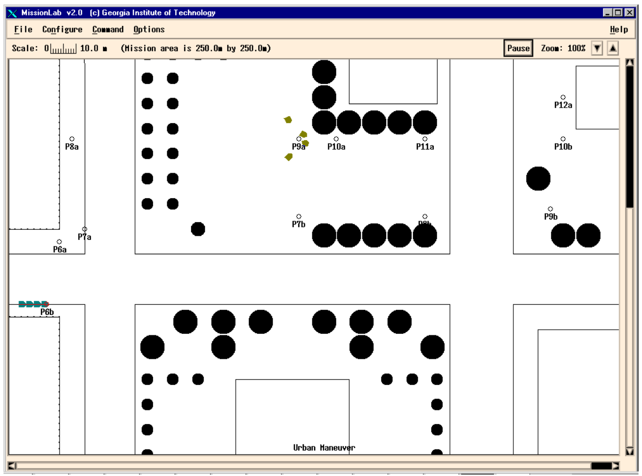

Figure 5: The MLab User Interface.

Figure 6 shows a high-level view of the RT-MLab architecture. The following section describes the RT-MLab functions in more detail.

Real-Time Analysis

RT-MLab analyzes the code that will be running on the robot using rate-monotonic scheduling theory to determine whether all processes can meet their timing constraints. We have replaced the MissionLab CNL-to-C compiler with a new compiler that interacts with MetaH to analyze mission feasibility, and then generates code that provides hard real-time performance guarantees when executed on a real-time operating system.

While the user's interaction with RT-MLab is essentially the same as the user's interaction with MissionLab, the robot executable has changed significantly. In MissionLab, each CNL node executes as a light-weight thread. In RT-MLab, sets of CNL nodes that run at the same frequency are grouped into a single MetaH *process*, as shown in Figure 7. At execution time, MetaH manages the execution of each process, and ensures that the dataflow requirements are met. Although MetaH supports multi-processor computation at both the analysis and execution stages of interaction, in this paper we assume that the robot has one processor.

There are two main advantages for setting different periods for different nodes. The first is *to accommodate different-rate sensors*. Assume, for example, that one sensor takes five seconds to update and process information, while another takes 100ms (say, vision and sonar, respectively). MetaH will run the faster sensor at every opportunity, while using old data from the slower sensor.

The second advantage is that *more frequent processes are treated with higher priority at run-time*. As a result, critical processes can interrupt less important processes, ensuring that performance guarantees are met. For example, even if a vision process overruns its allocated processor time, obstacle avoidance routines can still ensure that the robot will not crash. In particular, background processes will not affect the robot's execution performance. This feature allows some flexible scheduling of the processor, while ensuring that critical routines will be executed as frequently as necessary.

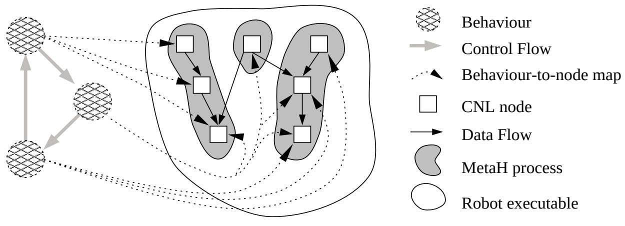

Figure 7: Behaviours, Nodes and Processes in RT-MLab. MetaH manages the inter-process interactions.

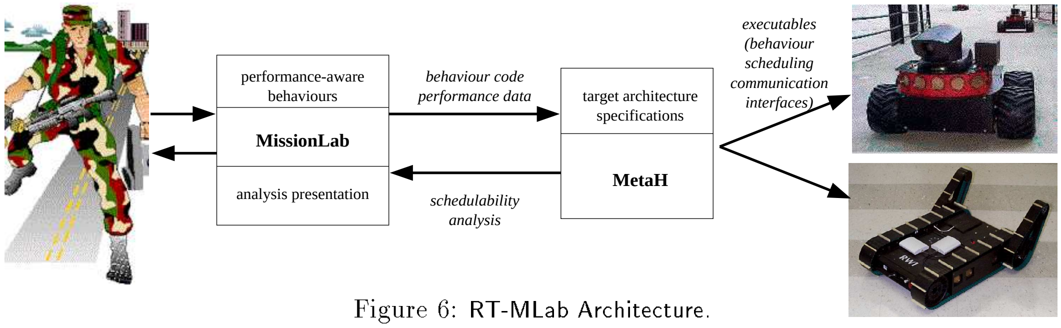

Figure 6: RT-MLab Architecture.

To perform real-time analysis, MetaH needs to know three things about the processes that will run in the final executable:

1. the dataflow between processes,
2. the execution time of each process, and
3. the period of each process.

The configuration network specifies a dataflow diagram between CNL nodes. From this network, the new CNL-to-C compiler builds the MetaH processes, and derives their execution times and periods.

Dataflow

As each behaviour configuration is compiled into CNL, the compiler also extracts the data flow between nodes. For example, in the configuration network shown in Figure 3, the node `detect_obstacle()` reads the sonar values and calculates an array of object locations that are used by `avoid_obstacle()` to calculate a desired travel vector.

This dataflow information serves two purposes. First, it allows us to derive appropriate periods for nodes that depend on inputs from different sources, as described below. From this derivation, the compiler can create MetaH processes that consist of all nodes running at the same period.

Second, by reasoning about the relative frequencies of processes that produce and consume data along particular dataflows, MetaH tools can automatically synthesize the necessary communication buffers to support over- and under-sampling. For example, if one process produces new data at a much slower rate than the consuming process runs, the data must be buffered to support oversampling.

Execution Times

Execution times need to be calibrated whenever hardware is changed on the robot, or when a new CNL node is added to the robot's capabilities. If a function relies on particular features of the hardware, its execution time may change. For example, if we change the frame grabber for a camera to a new frame grabber with higher resolution, the execution time of a vision routine may increase.

The execution time of a MetaH process is the sum of the execution time of each of its constituent nodes. The execution time of a node will remain constant for a given hardware configuration, and hence the execution time of a process can be directly calculated for each behaviour configuration.

Periods

To assess the feasibility of a behaviour configuration, RT-MLab requires that each process in the final executable have an assigned *period*. The period of a process is the length of time between consecutive dispatches of the process. The robot designer sets the period of a CNL node, and the CNL-to-C compiler places all nodes with the same period into a single MetaH process.

Many of the nodes in a robot program have periods that are hardware-dependent. For example, if the sonars are being fired ten times per second, then a node that acquires that sonar data should be run every 100 milliseconds. Since data flows along the arrows from node to node, the periods of upstream nodes can be used to derive suitable periods for downstream nodes that are not tied directly to hardware inputs. We want sensor data to flow as quickly as possible through the network, so we make downstream nodes run at least as frequently as their most frequent predecessor. The system designer needs to set the period for every node with no inputs (*critical node*).

For example, in the configuration network shown in Figure 3, the `detect_obstacle()` node should be run as frequently as the sonars can fire. The `get_location()` node, which finds the current robot location, is less critical than detecting obstacles, but its frequency determines how precisely the robot will be able to achieve a given goal location. The `extract_goal()` node, which gets the goal location from the GUI, can be run less frequently. Assume we assign the following periods:

`extract_goal()`	500 ms
`get_location()`	250 ms
`detect_obstacle()`	100 ms

The compiler can propagate these periods to downstream nodes, always selecting the more frequent value, generating:

`nav_at_goal()`	250 ms
`stop_robot()`	250 ms
`avoid_obstacle()`	100 ms
`navigate()`	100 ms
`move_robot()`	100 ms

The compiler creates the following three, independent MetaH processes:

500 ms:

```
extract_goal()
```

250 ms:

```
get_location()
nav_at_goal()
stop_robot()
```

100 ms:

```
detect_obstacle()
avoid_obstacle()
navigate()
move_robot()
```

MetaH will guarantee that each process uses the most recent information available from (possibly slower) processes that it depends on. For example, `navigate()` may be relying on 500ms-old data from `extract_goal()`. In a traditional, non-real-time architecture, communication delays or other problems may cause data to backlog, and a process might make decisions based on out-of-date information. MetaH eliminates the need to hand-manage data flow and latency concerns.

Configuration Analysis

The CNL-to-C compiler generates a set of MetaH process, along with their dataflow, execution times, and periods. RT-MLab uses this information to assess the feasibility of the user-level behaviour configuration. This "schedulability analysis" determines whether all processes can be executed at their specified periods. The analysis includes communication times between nodes, and overhead generated by the process dispatcher.

The analysis returns the total (or actual) utilization of the processor, and the breakdown (or maximum possible) utilization. For example:

```
Total utilization =   83.5%
Breakdown utilization =  96.7%
Processor schedule is feasible
```

The breakdown utilization is an indication of how disharmonic the process periods are. *Harmonic* periods occur when each period is an integer multiple of all lower periods; a set of harmonic processes is easy to schedule on the processor without leaving significant down-time. Dis-harmonic periods make it difficult to schedule high levels of utilization. For example, consider the two processes A and B depicted in Figure 8. Process A has an execution time of 2 ms, and a period of 5ms, yielding a processor utilization of 40%. Process B has an execution time of 3.25ms, and a period of 7ms, yielding a processor utilization of 46.4% utilization. However, despite the fact that they would only utilize 86.4% of the processor's total available time, there is no way for them to be scheduled to meet their timing constraints. In the current version of RT-MLab, the system designer is responsible for selecting harmonic periods.

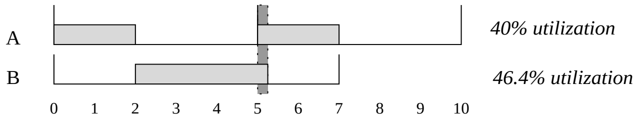

Figure 8: A set of unschedulable processes that do not overload the processor.

Example

We envision the end user exercising RT-MLab's real-time analysis tools in an interactive fashion. He would build an initial behaviour configuration, and ask RT-MLab to analyze it. If the configuration is not feasible, he would modify the configuration, iterating until he achieves a final, feasible configuration.

As a simple example, consider an exploring robot that hides at "home" whenever it senses an enemy robot.

In our first configuration, we use the "Near(Object)" behaviour to trigger the hide, and the "MoveTo(NamedLocation)" behaviour to get to the home location, as shown in Figure 9. The CNL network for this configuration contains 52 nodes.

MetaH uses our timing and period values to deter-

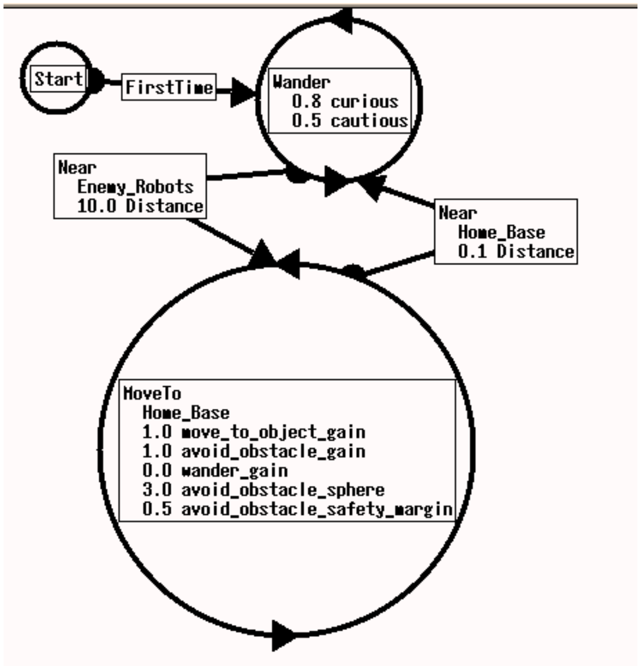

Figure 9: Exploring robot, using "Near" and "MoveTo". This behaviour configuration is not feasible.

mine the schedulability of this configuration, and reports that it is infeasible. One solution to this problem is to replace some of the expensive behaviours with similar cheaper behaviours.

The "Near(Object)" behaviour requires the robot to (i) calculate what objects are detected and where they are, and then (ii) calculate whether an enemy robot is one of the sensed objects, and finally (iii) decide whether the robot is "near." This procedure is extremely expensive, particularly when there are many objects in the environment. The behaviour "Detect(Object)," on the other hand, only requires the robot to calculate whether the enemy robot is detected, eliminating steps (ii) and (iii), and potentially significantly shortening step (i).

The "MoveTo(NamedLocation)" behaviour requires the robot to sense the home base before navigating there. If the home base doesn't move, then we can eliminate all the sensing steps by replacing this expensive behaviour with a simpler "GoTo(x,y)" behaviour.

By replacing both behaviours in the configuration, we get the configuration shown in Figure 10, which yields a CNL network of 39 nodes. This simpler configuration is feasible, while yielding the same general behaviour from the robot.

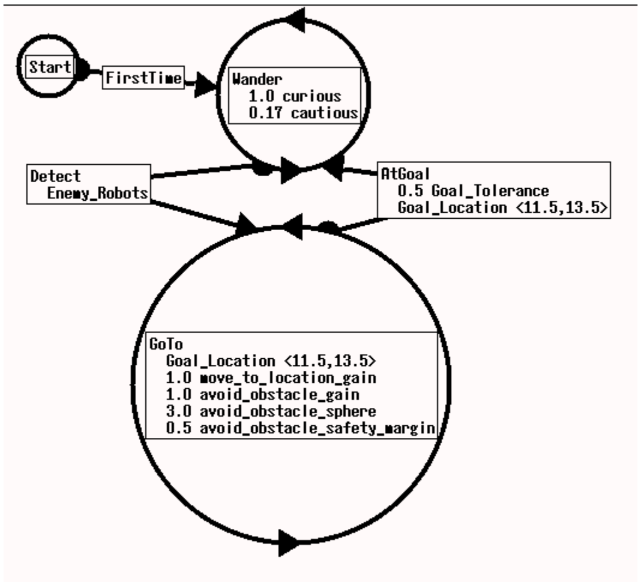

Figure 10: Exploring robot, using "Detect" and "GoTo". This behaviour configuration is feasible.

Future Work

Our future work will focus on increasing the usability of the RT-MLab's real-time analysis features. Our goal is to advise the end user through the process of creating a feasible configuration. We will add two main components:

1. a guarantee that user-settable parameters do not exceed robot-hardware limits (e.g. the robot will read sonar information frequently enough for its speed).
2. a search for alternate feasible configurations.

Hardware Constraints

In order to guarantee that user-settable parameters do not exceed hardware or environmental limits, we intend to create functions relating robot parameters to process periods. We will then check that these constraints are met. For example, we would like to guarantee that the robot processes its sonar data often enough that it doesn't bump into objects. (Note that this constraint is an extension of the current guarantee that the robot will read its sensors and react with a specified frequency.)

Let
$$\begin{aligned} v &= \text{robot speed} \\ p &= \text{sonar period} \\ a &= \text{robot acceleration} \\ d &= \text{maximum sonar visibility distance} \\ e &= \text{total execution time of all processes in configuration} \end{aligned}$$

Using these variables, we can define the following constraint:

$$\begin{aligned} d &>= v \times \text{reaction time} - \text{stopping distance} \\ d &>= v \times 2p - \frac{v^2}{2a} \end{aligned}$$

The reaction time of the robot is identified as shown in Figure 11. Note that the process `detect_obstacle()` will be *executed* once per period; **not** that it will *start execution* at exactly the start of the period. The maximum reaction time is hence $2p$.

This function computes the process periods directly from hardware parameters (e.g., sonar range) and user-specified behaviour parameters (e.g., robot speed).

These constraint functions will augment RT-MLab with two new abilities:

1. RT-MLab will guarantee constraints at a much more abstract level than the processor. RT-MLab currently guarantees that the processor will handle the load placed on it. These functions will allow RT-MLab to guarantee that the executable will meet certain behavioural performance requirements.
2. RT-MLab will be able to automatically deduce many of the process periods. Currently, the system designer sets a fixed period for each critical node. By using functions relating different hardware capabilities to each other, RT-MLab will be able to automatically calculate possible periods for sensor-based nodes.

Search for Feasible Configurations

There are two ways to make an infeasible configuration feasible. The first is to relax the user's constraints. The

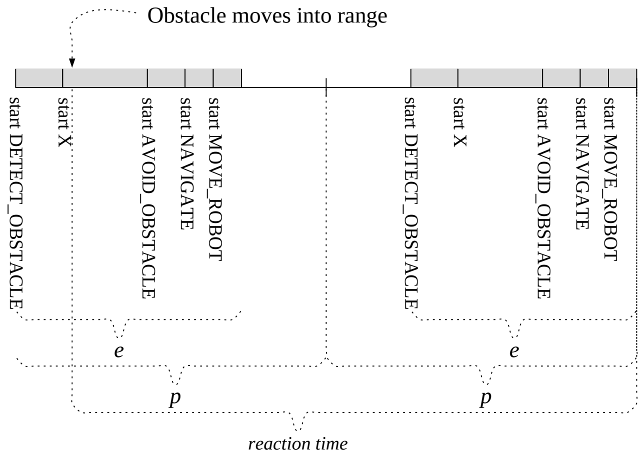

Figure 11: Reaction time for obstacle detection. The process "X" refers to any other nodes in the configuration network, and may be scattered throughout the execution.

second is to automatically swap behaviours for similar, cheaper behaviours. We are still considering possible designs for this second capability, and hence do not consider it here.

The constraints described above contain two types of variables: (1) hardware values, such as the maximum firing rate of the sonar card, or the execution time of a process, and (2) user variables, such as robot speed. Hardware values are fixed, but user variables can be relaxed. We can solve the constraint equations using the fixed values, yielding ranges on the variables.

We intend to use AI search techniques to search through the space defined by these values to yield feasible solutions, and present those with the least impact on performance to the user.

Related Work

While there are many reactive, behaviour-based architectures for robot control (e.g., RAPs, subsumption, 3T), very few have paid close attention to issues of hard-real-time responsiveness. Notable exceptions include Rex/Gapps, PRS, DR/MARUTI and CIRCA.

Rex is a language used to describe digital machines that can be viewed as reactive systems [Rosenschein & Kaelbling 1986]. Rex programs are compiled into automata descriptions that perform a constant-time mapping between inputs (sensors) and outputs (actuators). Gapps [Kaelbling & Rosenschein 1990] is a system for compiling declarative descriptions of agent behaviours into Rex machines. Thus Rex/Gapps resembles the original MissionLab system in providing a powerful and predictable reactive programming environment and executive. The improvements we made by adding real-time analysis and execution support would apply even more smoothly to Rex/Gapps, because of its formal fixed-time basis.

The Procedural Reasoning System (PRS) [Georgeff & Ingrand 1989; Ingrand & Georgeff 1990] has features making it suited to real-time applications. Ingrand and Georgeff have shown that, given certain assumptions about event frequencies and the form of the system's procedural knowledge, PRS can be guaranteed to notice (or begin reacting to) every world event within a bounded time. This guarantee is based on the fact that PRS processing is highly interruptible. However, "noticing" an event is distinguished from responding to the event. It is also possible to limit PRS' inferencing capabilities and make guarantees about overall response time [Ingrand & Georgeff 1990]. However, PRS has not been tied to real-time operating systems or analysis mechanisms, so these potential performance guarantees remain unrealized.

Hendler and Agrawala [1990] did preliminary work integrating an enhanced Dynamic Reaction (DR) system and the MARUTI operating system to implement guaranteed real-time reactive reasoning that closely resembles the output of RT-MLab, without the automatic analysis and programming environments. The DR system sets up asynchronous monitor processes to check conditions on specific world model features: signals from these monitors drive changes in reactive activities. The MARUTI operating system provides explicit support for scheduling hard real-time tasks on distributed systems, guaranteeing the execution of jobs that are accepted. By using MARUTI to schedule and execute the reactive elements of DR, the combined system can make performance guarantees. DR/MARUTI does not have mechanisms to automatically reason about or analyze its scheduling requirements.

The CIRCA system [Musliner, Durfee, & Shin 1993; 1995] combines a hard-real-time reactive executive with soft-real-time AI planning methods that automatically generate reactive plans given a set of high-level goals. CIRCA includes its own scheduling module that allows it to reason explicitly about the real-time requirements of its reactive behaviours, and automatically adjust those behaviours (plans) when its execution resources are not sufficient. In many ways, CIRCA resembles RT-MLab with the user and GUI replaced by AI planning methods. We are anxious to explore the potential for combining CIRCA's planning capabilities with the alternative execution semantics offered by RT-Mlab.

Summary

Our RT-MLab extensions to MissionLab provide real-time schedulability analysis and reliable real-time execution support for robot behaviour configurations. Using formal real-time analysis and execution tools ensures that the robot programs created in RT-MLab will be executed as designed, without unpredictable timing behaviour or communications delays. As one of the first robot control architectures providing hard real-time guarantees, we believe that RT-MLab is a major step forward in the deployment of reliable robotic systems.

Acknowledgments

The authors would like to thank Ron Arkin, Tom Collins, Tucker Balch, and Steve Vestal for their help and insightful comments.

References

[Binns & Vestal 1993] Binns, P., and Vestal, S. 1993. Scheduling and communication in MetaH. In *Real-Time Systems Symposium.*

[Endo *et al.* 1999] Endo, Y.; MacKenzie, D. C.; Ali, K. S.; Balch, T.; Cameron, J. M.; and Cheng, Z. 1999. MissionLab: User manual for MissionLab version 3.0. Technical report, College of Computing, Georgia Institute of Technology, Atlanta, GA. Available via http://www.cc.gatech.edu/ai/robot-lab/research/MissionLab/mlab_manual.ps.gz.

[Georgeff & Ingrand 1989] Georgeff, M. P., and Ingrand, F. F. 1989. Decision-making in an embedded reasoning system. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 972–978.

[Hendler & Agrawala 1990] Hendler, J., and Agrawala, A. 1990. Mission critical planning: AI on the MARUTI real-time operating system. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, 77–84.

[Ingrand & Georgeff 1990] Ingrand, F. F., and Georgeff, M. P. 1990. Managing deliberation and reasoning in real-time AI systems. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control*, 284–291.

[Kaelbling & Rosenschein 1990] Kaelbling, L. P., and Rosenschein, S. J. 1990. Action and planning in embedded agents. In *Robotics and Autonomous Systems 6*, 35–48.

[Krishna & Shin 1997] Krishna, C. M., and Shin, K. G. 1997. *Real-Time Systems.* McGraw Hill.

[MacKenzie, Arkin, & Cameron 1997] MacKenzie, D. C.; Arkin, R. C.; and Cameron, J. M. 1997. Multi-agent mission specification and execution. *Autonomous Robots* 4(1):29–52.

[Musliner, Durfee, & Shin 1993] Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1993. CIRCA: A cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man, and Cybernetics* 23(6):1561–1574.

[Musliner, Durfee, & Shin 1995] Musliner, D. J.; Durfee, E. H.; and Shin, K. G. 1995. World modeling for the dynamic construction of real-time control plans. *Artificial Intelligence* 74(1):83–127.

[Rosenschein & Kaelbling 1986] Rosenschein, S. J., and Kaelbling, L. P. 1986. The synthesis of digital machines with provable epistemic properties. In *Proceedings of the Conference on Theoretical Aspects of Reasoning About Knowledge*, 83–98.

[Shin & Ramanathan 1994] Shin, K. G., and Ramanathan, P. 1994. Real-time computing: A new discipline of computer science and engineering. *Proceedings of the IEEE* 82(1):6–24.

[Stankovic 1988] Stankovic, J. A. 1988. Misconceptions about real-time computing: A serious problem for next-generation systems. *IEEE Computer* 21(10):10–19.

[Vestal 1997] Vestal, S. 1997. An architectural approach for integrating real-time systems. In *Workshop on Languages, Compilers and Tools for Real-Time Systems.*

[Vestal 1998] Vestal, S. 1998. MetaH user's manual. Available from http://www.htc.honeywell.com/-metah/uguide.pdf.