Transparent, Flexible, and Resource-adaptive Image Processing for Autonomous Service Robots [1]

Michael Beetz, Tom Arbuckle, Armin B. Cremers, and Markus Mann [2]

Abstract. This paper describes SRIPPs, structured reactive image processing plans, that are tailored for the needs of autonomous service robots. SRIPPs are implemented on top of a modular, dynamically configurable, architecture called RECIPE and specified as subplans written in the robot control/plan language RPL [13]. Specifying image processing routines transparently and explicitly as part of the robots' plans rather than hiding them in separate modules makes the robots' visual capabilities more effective, efficient, and robust. The use of SRIPPs enables robots to generate, reason about and revise their image processing routines. They can also synchronize their image processing activities with their other actions and use control structures to recover from image processing failures.

1 Introduction

Many autonomous service robots that act in dynamically changing environments need to acquire and interpret various kinds of visual information. Consider an office delivery robot that is to accomplish the following job: "Get the red envelope from Wolfram's desk." Such a job specification contains visual descriptions of the objects to be delivered as well as the locations for collection and delivery of the objects. To achieve the task, the robot has to perform different kinds of visual tasks like recognizing pieces of furniture or parts thereof, visually searching regions in images, detecting people, and so on.

Making their image processing (IP) capabilities efficient and reliable often requires robots to (1) use contextual information to simplify IP tasks, (2) tailor IP routines to specific situations, (3) tightly integrate the IP capabilities into the operation of the control system, and (4) make optimal use of the robot's computational resources. In particular, we consider the following features to be essential for successfully integrating IP capabilities into robot control systems.

- **Stability.** The robot control system should function stably, even in the case of some system failure.
- **Resource management.** IP capabilities should make efficient use of the limited facilities available to the robot such as cameras and the processing, memory and communication resources. Employing these resources optimally requires that the strategy for managing IP resources be adaptive.
- **Distributed control.** The robot's control system is physically distributed on different computers. Generally the bandwidth for communication is limited and the information transferred within the system should be as concise as possible.
- **Image processing routines as actions.** IP routines are activated, controlled, and synchronized by the robot control system. Like other actions, visual routines have both desired effects, such as the acquisition of new information, and side effects, such as pointing the camera or consuming computational resources.
- **Image processing routines as plans.** The robot reformulates its IP tasks using local information it gathers from its sensors and adjusts the way it solves those tasks depending on this information.
- **Image processing routines in context.** The IP tasks of autonomous robots are embedded in complex activities. Therefore, visual routines need not always work. Often it is better to apply methods that, with a high probability, return good results in conjunction with methods for failure detection [10] and repair [12].

We present the design of a programming system for IP routines which satisfies the requirements above. Our solution consists of

- **RECIPE**, a dynamically loadable, modular architecture in a distributed robot control system that provides the basic IP functionality and manages images and other IP data structures. It provides a variety of standard IP routines such as edge detectors, convolutions, noise reduction, segmentation, etc.
- **RPL$_{\mathbf{IP}}$**, an extension of the abstract machine provided by the robot control/plan language RPL. RPL$_{\mathrm{IP}}$ provides suitable abstractions for images, regions of interest, etc, and supports a tight integration of the vision routines into the robot control system.
- **Image Processing Plans** that provide various methods for combining IP methods into IP pipelines. IP plans support the implementation of robust vision routines and the integration of other sensors such as laser range finders and sonars for object recognition tasks and scene analysis. Since vision routines are RPL programs, they can be constructed, revised, and reasoned about while the robot control program is being executed.

The most significant contribution of this paper is the idea of having a single language to control physical actions, perception actions, and the internal operations of IP routines. This uniform framework views IP routines as actions that need to be controlled, monitored, and synchronized and plans that can be generated, reasoned about, and revised. These features are crucial constituents of effective, efficient, and robust image processing for service robot applications.

Before going on to describe the structure of the remainder of the paper, we give a short example in which the need for these requirements is demonstrated.

2 Image Processing as Part of Robot Control

We have implemented SRIPPs as part of the FAXBOT control system, a robot control system that is designed for robust and efficient execution of delivery plans on the autonomous mobile robot RHINO (see Fig. 1), an RWI B21 robot. RHINO is equipped with three PCs

[1] This work has partially been supported by EC Contract No ERBFMRX-CT96-0049 under the TMR Programme.

[2] University of Bonn, Dept. of Computer Science III, Roemerstr. 164, D-53117 Bonn, Germany. email: beetz, arbuckle, abc, mann@cs.uni-bonn.de

ECAI 98. *13th European Conference on Artificial Intelligence*
Edited by Henri Prade
Published in 1998 by John Wiley & Sons, Ltd.

connected to a computer network via a tetherless radio link. The sensor system consists of 24 ultra-sonic proximity sensors, two laser range finders, and a CCD stereo color camera system.

Fig. 1: RHINO

FAXBOT operates in a part of an office building (see Figure 2) and uses a symbolically annotated 3D world model that includes items such as walls, doorways, rooms and static pieces of furniture. The world model provides all the information required for fast and collision free navigation and accurate robot localization [17]. The robot possesses a navigation system that can determine the robot's position in this environment with an average position accuracy of about eight centimeters and an average orientational error of less than two degrees [17].

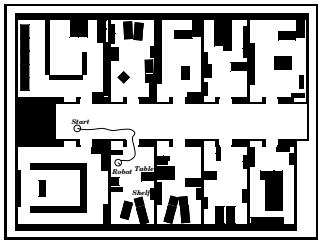

Fig. 2: *FAXBOT navigates in front of a shelf, turns to face the shelf, and runs the routines for recognizing the shelf.*

Let us consider an example showing the advantages accruing from having image processing as an integral, explicit and transparent part of the control system. We consider the classic task of visual place recognition and the recognition of known pieces of furniture within an office environment. In this problem context, specialization and adaptation to the environment promise great simplifications of the visual processing required. The IP functionality for this type of task is well known, permitting us to focus on the added functionality which comes from employing SRIPPs. Simply put, FAXBOT employs a geometric projection algorithm that takes RHINO's position and the direction in which the camera is pointing as arguments, retrieves the furniture items in the robot's view, and predicts where the corners of the furniture item should appear in the camera image. Thus FAXBOT applies a compound IP routine which searches for the predicted lines suggested by the geometric projection algorithm.

FAXBOT is responsible for the control of the IP operations required for object recognition. Since these operations are represented as plans, FAXBOT can tailor the visual recognition of furniture items by selecting the lines to be found and by sorting these lines in an order that should promise fast and reliable recognition. As an example of specialization, suppose that the color of the item of furniture differs significantly from the background. The object recognition procedure can then look for the boundary corners first. If, for example, proximity sensors detect an object standing in front of the item of furniture, the process can adapt its plan, and thus its sequence of IP operations, to prefer vertical lines and high corners. The procedure will run the routines for detecting and recognizing lines in parallel or sequentially depending on CPU load. The confidence value for a successful recognition will also be set according to specified deadlines for the recognition process. In the case of recognition failures the robot control program running the recognition procedure can try to relocalize the robot or move the robot to a different location near the furniture item to perhaps get a better view.

The following figures show the output of different steps of the recognition procedure. In Figure 3 one can see the region of interest (black) that is computed from the degree of confidence in the robot's location with the white lines being the edges detected by the edge detector within this region. Figure 4 shows the final result of recognizing the shelf. The black lines are the predicted ones and the white ones are the recognized lines. Such results are to be expected from this type of IP.

It would be impossible to achieve this level of flexibility and robustness if image processing were hidden within discrete and fixed IP modules that could be used only in a very limited and predefined way and were controlled by stored sequences of IP requests. Since IP routines in our approach are dynamically generated and revised during robot operation, the robot control system needs direct access to the visual information sensed by the robot. Moreover, since the control routines are concurrent programs, they may simultaneously need to read or modify the shared visual information. In this context, the IP facility must enforce controlled access to the visual information to ensure stability and optimal use of available resources.

Fig. 3. *Edge detection step.*

Fig. 4. *Recognizing the shelf.*

The remainder of the paper is organized as follows. Section 3 describes the interface between the IP facilities and the plan language RPL. Section 4 shows how SRIPPs, plans that specify IP routines, are implemented. We demonstrate the modularity and transparency of SRIPPs and that they can be generated and revised by automatic planning techniques. Section 5 describes how the requirements for stability, resource management and distributed control are met by RECIPE, a runtime configurable, image processing system. Finally, we summarize our results, giving directions for future research.

3 Robot Control and the IP Interface

To explain how the IP operations are embedded within the robot control system, we must first explain how the robot control system itself functions as well as the interface between the plans and the IP operations. IP routines are run as parts of structured reactive controllers (SRCs). SRCs specify how the robot is to respond to events and feedback generated by the low-level control modules of the robot. The main components of SRCs are the *fluents*, the *process modules* and the *structured reactive plan* [5].

Fluents are registers or program variables that signal changes of their values. They are used to store events, sensor reports and feedback generated by low-level control modules. Moreover, since fluents can be set by sensing processes, physical control routines or by assignment statements, they are also used to trigger and guard the execution of high-level control routines.

For image processing we introduce new categories of fluents that store basic data structures such as images, regions of interest, lines, segments, confidence values, or their indices.

Process modules are elementary program units that constitute a uniform interface between "high-level" plans and the "low-level" continuous control processes, such as IP routines.

The process module GRAB-IMAGE, for example, which is provided in our IP extension of RPL is activated with a camera name, image size, and whether the image is to be color or grayscale as its parameters. The module updates the fluents IMAGE, in which the grabbed image is stored, and DONE?, which is pulsed upon process completion.

Another process module, PROCESS-IMAGE, takes an image, the region of interest, the IP operator, and additional parameters for tuning as its arguments. The IP operators include edge detectors (Sobel, Canny, ...), line followers, segmentation operators, and so on. The image and region of interest parameters are the indices of the images and regions of interest to be found in the image and ROI tables of the RECIPE module. The fluents that are updated by the PROCESS-IMAGE module are the IMAGE, ROI, and DONE fluents that contain the image resulting from the application of the operator and the done signal.

There are also process modules for matching lines, generating regions of interest, segmenting images, displaying images, and so on.

Structured Reactive Plans (SRPs), high-level control routines, are concurrent, reactive control programs. Executing an SRP activates, deactivates, and parameterizes process modules. To realize structured reactive image processing plans (SRIPPs), an obvious extension of SRPs, we have augmented the interface between the high-level control language and the low-level control processes.

RPL and Image Processing Control Structures. The robot's plans and IP routines are implemented in RPL (Reactive Plan Language) [13]. RPL provides conditionals, loops, program variables, processes, and subroutines as well as high-level constructs (interrupts, monitors) for synchronizing parallel actions. To make plans reactive and robust, it incorporates sensing and monitoring actions, and reactions triggered by observed events.

Moreover RPL provides concepts that are crucial for the realization and integration of IP routines. The PLAN-statement has the form (PLAN *STEPS CONSTRAINTS*). *STEP*s have the form (:TAG *NAME SUBPLAN*) and tag *SUBPLAN* with the name *NAME*. *constraint*s have the form (:ORDER S_1 S_2) where S_is are name tags of the plan steps. *Steps* are executed in parallel except when they are constrained otherwise. TRY-ALL, another RPL control structure, can be used to run alternative methods in parallel. The compound statement succeeds if one of the methods succeeds. Upon success, the remaining processes are terminated.

An Example Program. There follows a program segment that grabs a 240 × 320 pixel image with the left camera, returning an index for the image. The program segment creates two local fluents IMG-ID-FL and DONE-FL. It passes the fluents as call parameters to the process module GRAB-IMAGE. The module sets IMG-ID-FL to the index of the grabbed image and pulses the fluent DONE-FL upon completion.

```
(WITH-FLUENTS (IMG-ID-FL DONE-FL)
   (GRAB-IMAGE :CAMERA          :LEFT
               :SIZE            (240 320)
               :COLOR           :TRUE
               :GRABBED-IMAGE   IMG-ID-FL
               :DONE            DONE-FL)
   (WAIT-FOR DONE-FL)
   (FLUENT-VALUE IMG-ID-FL))
```

The segment activates the module GRAB-IMAGE, waits for the image capture to complete and returns the value of the fluent IMAGE-ID-FL.

4 Structured Reactive Image Processing Plans

To reason about IP routines, they have to be implemented as plans. Plans are control programs that can not only be executed but also automatically analyzed and transformed. IP pipelines are an appropriate means of representing complex IP routines as plans because pipelines are modular, transparent and make the sequencing of IP operations explicit [16]. Image processing pipelines are data flow programs, directed graphs, in which each node represents an IP operation and each directed arc represents a path over which data such as images, regions of images, or extracted lines, flow.

Figure 5 shows an image processing pipeline for recognizing a corner of a known item of furniture as discussed in section 2. The pipeline has to be provided with an image captured by the camera, the predicted coordinates of the corner in the image, and a region of the image in which the line should occur considering the current uncertainty about the robot's own position. These inputs of the pipeline have the names PREDICTED-LINE, ROI-SPEC, and IMAGE. The pipeline is a sequence of four operations. The first one, CREATE-ROI, extracts the region specified by ROI-SPEC from the image and provides the region named ROI as its outputs. The second pipeline step applies an edge detector (called Sobel) to the region and this region is then processed by a line following algorithm. Finally, RECOGNIZE-LINE takes the region with the lines and compares these candidate lines with the predicted line provided as a pipeline input. The best match found by the last pipeline step, and the confidence that this line is the image of the predicted line, are the outputs of the pipeline.

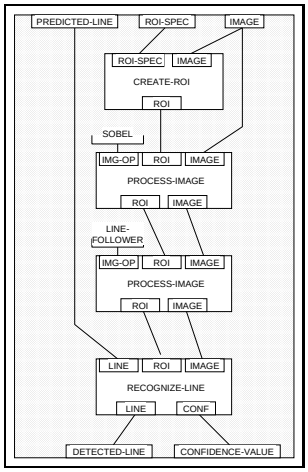

Fig. 5: *Image processing pipeline.*

Image processing pipelines as RPL plans. IP pipelines are specified using the macro IMAGE-PROCESSING-PIPELINE, which has the form

```
(IMAGE-PROCESSING-PIPELINE [Pipeline Interface]
                           [Pipeline Steps]
                           :CONNECTIONS [Pipeline Connections])
```

where *Pipeline Interface* specifies the input and output data paths, *Pipeline Steps* the IP operators, and the *Pipeline Connections* are the data paths between the pipeline steps. The macro IMAGE-PROCESSING-PIPELINE is expanded into an RPL plan that contains a WITH-FLUENTS-statement and PLAN-statements. Connections are implemented as shared fluents declared in the WITH-FLUENTS statement: one IP operator writes the fluent that the next one takes as an input. Each plan step starts the IP operation and waits for its completion. The connections are then transformed into :ORDER constraints for the PLAN statement so that a pipeline step can run only if all steps that produce the step's inputs have been completed.

Image processing details are hidden by defining a procedure that takes the input and output fluents of the pipeline as arguments.

```
(DEF-INTERP-PROC FIND-A-PREDICTED-LINE
                 (PRED-LINE-FL ROI-SPEC-FL IMG-FL
                  DETECTED-LINE-FL CONFIDENCE-FL DONE?)
   [Image Processing Pipeline])
```

Controlling image processing routines. The following piece of code shows the synchronization of IP routines with the rest of the robot control system.

```
(WITH-VALVE WHEELS
   (WITH-VALVE CAMERA
      (TRY-ALL
         (WAIT-TIME 5)
         (WAIT-FOR (> CONFIDENCE-FL 0.7))
         (SEQ [(FIND-A-PREDICTED-LINE PRED-LINE-FL ... DONE?)]
              (WAIT-FOR DONE?)))))
```

To avoid destructive cross-process interferences, no other process should be able to make the robot leave its current location or redirect the camera. This is done using *valves*, semaphores that processes must own in order to issue particular kinds of commands. Thus, any process that causes the robot to move to another place must first request the valve WHEELS, move the robot upon receiving the valve, and release the valve after the completion of the movement.

Another important aspect of controlling IP routines is the assignment of computational resources to IP tasks. In the example above we use the TRY-ALL statement to do this. The TRY-ALL statement succeeds if five seconds have passed, the confidence fluent exceeds the confidence threshold of 0.7, or if the FIND-A-PREDICTED-LINE routine has completed. Thus, the code segment above runs the FIND-A-PREDICTED-LINE IP pipeline until there is enough evidence for a correctly recognized object and at most for five seconds. The valves WHEELS and CAMERA ensure that the robot and the camera are not moved.

Application of alternative methods. We can apply RPL control structures in order to combine alternative IP methods, for example, different methods for recognizing a piece of furniture. If, for instance, the item has a distinctive color, the robot can first extract similarly colored segments and then apply the other operations only to these segments. The different methods can be applied in parallel or sequentially, until one method acquires enough evidence or the accumulated evidence of all methods surpasses a given threshold. This can be specified using such RPL control structures as TRY-ALL.

Failure recovery. Robust control systems must deal with failures of their IP routines. Consider the routine FIND-A-PREDICTED-LINE which will fail if the furniture has been moved, if the robot is uncertain about its position, if the piece of furniture is (partly) occluded, and so on. FAXBOT's plans that execute the IP routines try to detect if the routines fail and signal failures in these situations. Thus, FIND-A-PREDICTED-LINE signals a failure if the predicted line could not be recognized or could be recognized only with a low confidence. In the case of failure, robust control structures can then check the robot's confidence in its position estimate or use laser and sonar range finders to detect possible occlusions and recover from recognition failures.

Tailoring pipelines. There are many IP parameters that the robot can adjust by sensing but cannot predetermine. For example, the quality of lighting within a room can be perceived by the robot's sensors but the robot cannot know in advance whether a room will be adequately lit. Thus, the IP pipelines should be tailored to fit the robot's particular circumstances and environment. Instead of requiring the IP routines to have general applicability and reliability, FAXBOT applies a selection of context-specific IP routines that work well in the contexts for which they have been tailored.

It is possible to dynamically modify the IP pipelines because SRIPPs are plans and because specific tools provided by RPL permit processes to (1) project what might happen when a robot controller executes a SRIPP [14]; (2) infer what might be wrong with a SRIPP given a projected execution scenario; and (3) perform complex revisions on SRIPPs [14, 6].[3] Thus, explicitly representing IP routines in the robot's plan enables the robot to apply various methods to simplify its vision tasks. It can pose more specialized vision problems [11] and adapt the computer vision strategies to the particular context (environment, task, image) during execution [15]. Mechanisms for revising plans during their execution are described in [7].

5 RECIPE — A Modular IP System

RECIPE – Reconfigurable Extensible Capture and Image Processing Environment – is the system module responsible for the capture and processing of the visual information controlled and modeled by the SRIPPs. It has two principal components - a capture server and several copies of an image processing (IP) server.

The capture server is a lightweight, robust process which has three main responsibilities. It stores the images captured by the system's cameras. It also stores all important system data and configuration information for RECIPE. On many supported architectures, stored information is stored persistently thus making a system restart possible even in the case of a total system failure. Finally, the capture server acts as a watchdog for all IP servers, restarting and reconfiguring them if their processing should cause them to hang or crash.

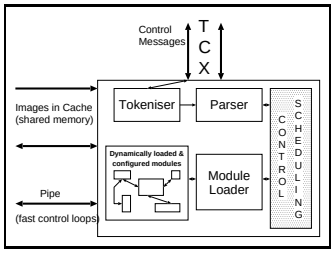

Fig. 6: *IP Server Design.*

The IP server forms the heart of the RECIPE system. It is a script controlled modular architecture for the implementation of image processing operations and pipelines. The IP server loads and unloads image processing functionality according to commands sent by the controlling SRIPPs. Communication with the capture server is established by way of shared memory (especially for images) and by pipes/sockets (for fast control and configuration information). An IP server is shown in figure 6.

RECIPE has been designed to satisfy many of our requirements for stability, distributed control and resource management. The capture server is a small, lightweight process with limited and clearly defined functionality which additionally acts as a system watchdog. It restarts and reconfigures IP servers in the event of a partial system failure, thus giving the system greater tolerance of badly implemented IP modules. Moreover, it stores all relevant system status, all images and all ROIs in a central (usually persistent) location.

As for the criterion of distributed control, RECIPE performs all of its IP on the robot. Its run-time configuration is dependent on the robot's context and only brief control messages need be transferred between the external controlling processes and the RECIPE system. RECIPE therefore encourages distributed control.

The management of available resources is one of RECIPE's essential features. Since IP operations are largely written to be as efficient as possible, further optimizations in system efficiency can only be made by employing context specific algorithms for problem solving. RECIPE's modules are loaded and unloaded on demand permitting such contextual implementation. Moreover, by employing dynamic linking RECIPE both restricts the memory footprint of the process and reduces complexity of the system's code.

6 Related Work

In contrast with other IP systems such as TargetJr, Khoros or Matlab, at any one moment RECIPE only provides a restricted and dense subset of IP operations tailored to the task at hand. It manages the resources available to the robot, loading and unloading functionality on demand and monitoring its operational status. We consider RECIPE to provide a subset of the functionality offered by other IP systems in a manner suited for operation on an autonomous robot.

Research in the active vision area [2, 3, 4] focuses on increasing the performance and robustness of IP routines by selectively applying special purpose routines that extract purposive information from the image. SRIPPs and RECIPE focus on particular aspects of active vision, namely how to specify subplans that can perform active IP in a robot control/plan language and the resource management facilities necessary for robust performance under scarce resource constraints. Horswill studies specialization methods for deriving fast IP methods from environmental constraints [11].

Ullman's theory of visual routines [18] proposes that our visual perception is achieved by modular visual routines. These are com-

[3] These facilities have been fully implemented for the physical actions of the robot but are not yet fully applicable to the image processing routines.

posed of simple visual actions using a small set of registers containing visual data. In conjunction with fluents, SRIPPs can analogously be considered to be an expressive high-level programming language for the implementation of vision routines.

RECIPE in conjuction with RPL and SRIPPs has much in common with the Gargoyle system[15] embedded in a three-tiered robot control architecture [8]. Gargoyle is also a modular platform-independent IP environment that can assemble and execute IP pipelines. As with RECIPE, Gargoyle has an interface to a high level robot control system but in the case of Gargoyle, the IP routines are not plans the control system can reason about. RECIPE's two component design in which one robust, lightweight component acts as a system watchdog, restarting and reconfiguring failed processes emphasizes RECIPE's focus on overall system robustness. RECIPE is more specialized for the role of an IP system running on a robot. Its modular architecture and run-time configurability give it a greater configurability than Gargoyle and RECIPE is able to use this to good effect in conjunction with the RPL SRIPPs. There is a number of robot control systems that tightly integrate IP modules eg. [1] and [9]. To the best of our knowledge, none of these systems are concerned with reasoning about IP routines at a symbolic level.

7 Discussion and Conclusions

Discussion. In this paper, we have only discussed the task of place recognition where processing delays are tolerable. In the version of the robot office courier we have presented, hard real-time tasks, such as collision avoidance and localization, are handled by sensor interpretation systems which do not employ IP [17]. In our current research we are also investigating IP tasks such as people detection, gesture detection and recognition which have firm scheduling requirements. These scheduling requirements are met by fast feedback loops (sending asynchronous signals) which are handled by RPL.

Concerning the topic of robustness, we have been concerned with fault tolerance of IP routines and the programs that control them rather than fault avoidance. Therefore we did not directly address the robustness of the IP algorithms themselves. Instead our aim has been to ensure that the IP algorithms and routines are capable of detecting their own success and failure so that the high-level control system can specify appropriate control structures to recover from failures.

Similarly, we have not been concerned with how the IP algorithms handle uncertainty in the manner in which they evaluate the likelihood of their results. Rather we have sought to exploit the robot's knowledge of its uncertainty in its state estimation – some function of the probability distribution for the state estimate provided by our localization routines – to select and parameterize IP algorithms[5]. The case that this distribution is multi-modal can be handled appropriately by RPL's ability to speculatively and concurrently execute different IP methods or parameterizations. Of course uncertainty might give rise to incorrect or ambiguous IP results. In most cases this will cause the planned actions to fail and the robot will then perceive such failures and handle them correctly.

Finally, within complex service applications there is a need to be able to reason about goal sequencing and plan step scheduling. Only by using a high-level plan language such as RPL can we perform the necessary transparent embedding of IP within the control system. We have developed similar embeddings for natural language understanding and for action planning [7].

Conclusions. This research has shown how the processes of a general robot control system can be combined with IP in a transparent way given a suitable protocol and the support of control abstractions in the high level language. SRIPPs represent an abstraction of low-level image processing (IP) operations into the same domain as the high-level robot control language. Having IP operations as an integral part of the control language provides additional flexibility and prevents the artificial distinction between IP and other sensing actions of the robot. In addition, the explicit, transparent and modular representation of IP routines allows for the application of AI plan generation and plan revision techniques to IP tasks. The implementation of IP routines must be supported by an underlying plan language which provides concurrency and event driven execution.

The flexible and robust execution of IP routines on autonomous mobile robots would not be possible without the facilities provided by some dynamically configurable IP engine. The RECIPE system fulfills all of the above requirements as well as providing additional system robustness, flexibility and run-time resource management.

In our future research, we will focus on the automatic planning and learning of IP operations depending on robot context that exploit the modular architecture and processing abilities provided by SRIPPs and RECIPE.

REFERENCES

[1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand, 'An architecture for autonomy', *Int. J. of Robotics Research*, (1998).

[2] J. Aloimonos, I. Weiss, and A. Bandopadhay, 'Active vision', *International Journal on Computer Vision*, 333–356, (1987).

[3] R. Bajcsy, 'Active perception', *Proc. IEEE*, **76**, 996–1005, (1988).

[4] D. Ballard, 'Animate vision', *Artificial Intelligence*, **48**, 57–86, (1991).

[5] M. Beetz, W. Burgard, D. Fox, and A. Cremers, 'Integrating active localization into high-level control systems', *submitted to Robotics and Autonomous Systems*, (1998).

[6] M. Beetz and D. McDermott, 'Improving robot plans during their execution', in *Second International Conference on AI Planning Systems*, ed., Kris Hammond, pp. 3–12, Morgan Kaufmann, (1994).

[7] M. Beetz and D. McDermott, 'Local planning of ongoing behavior', in *Third International Conference on AI Planning Systems*, ed., Brian Drabble, pp. 3–12, Morgan Kaufmann, (1996).

[8] P. Bonasso, J. Firby, E. Gat, D. Kortenkamp, D. Miller, and M. Slack, 'Experiences with an architecture for intelligent, reactive agents', *J. Experimental and Theoretical Artificial Intelligence*, **9**(1), (1997).

[9] J. Crowley and H. Christensen, *Vision as Process*, Springer, 1995.

[10] B. Donald, 'Planning multi-step error detection and recovery strategies', *Int. J. Robotics Research*, **9**(1), 3–60, (1990).

[11] I. Horswill, 'Analysis of adaptation and environment', *Artificial Intelligence*, **73**, 1–30, (1995).

[12] S. Kannan and M. Blum, 'Designing programs that check their work', in *Proc. 1989 Symposium on Theory of Computing*, (1989).

[13] D. McDermott, 'A reactive plan language', Research Report YALEU/DCS/RR-864, Yale University, (1991).

[14] D. McDermott, 'Transformational planning of reactive behavior', Research Report YALEU/DCS/RR-941, Yale University, (1992).

[15] Peter N. Prokopowicz, Michael J. Swain, R. James Firby, and Roger E. Kahn, 'GARGOYLE: An environment for real-time, context-sensitive active vision', in *Proc. of the Fourteenth National Conference on Artificial Intelligence*, pp. 930–937.

[16] Rasure and Young, 'An open environment for image processing software development', in *Proceedings of the SPIE/IS&T Symposium on Electronic Imaging (SPIE-92)*, (February 1992).

[17] S. Thrun, A. Bücken, W. Burgard, D. Fox, T. Fröhlinghaus, D. Hennig, T. Hofmann, M. Krell, and T. Schimdt, 'Map learning and high-speed navigation in RHINO', in *AI-based Mobile Robots: Case studies of successful robot systems*, eds., D. Kortenkamp, R.P. Bonasso, and R. Murphy, MIT Press, Cambridge, MA, (1998). to appear.

[18] S. Ullman, 'Visual routines', *Cognition*, 97–159, (1984).