# Integrating Reaction Plans and Layered Competences Through Synchronous Control

R. Peter Bonasso

The Autonomous Systems Laboratory
The MITRE Corporation
7525 Colshire Drive, Mclean, Virginia 22102
pbonasso@mitre.org

## Abstract

This paper describes an agent architecture and its implementation for situated robot control in field environments. The architecture draws from the ideas of universal plans and subsumption's layered control, producing reaction plans that exploit low-level competences as operators. The architecture has been implemented in an extended version of the GAPPS/Rex situated automata programming language. This language produces synchronous virtual circuits which have been shown to have formal epistemic properties. The resulting architecture exhibits robust task execution, has high-level goal representations, and maintains consistent semantics between agent states and the environment. Ongoing experiments using the architecture with two land mobile robots and one undersea mobile robot are described. The robots perform their tasks robustly during normal changes in the task environments.

## 1 Introduction

We are interested in programming robots to carry out tasks robustly in field environments. Field environments are those in which events for which the robot has a response can occur unpredictably, and wherein the locations of objects and other agents is usually not known with certainty until the robot is carrying out the required task. We expect the agent to be able to deal with its own mechanical and sensor limitations (e.g., wheel slippage, limited sensor sampling rates), and with natural changes in the flow of events (e.g., normally moving obstacles or other agents, transition from day to night). But when confronted with events for which it has no response (e.g., a meteor shower or runaway train), we expect the agent only to safely cease operations.

As researchers began to realize that traditional AI planning techniques produced robot plans which failed in field environments, a number of new approaches to robot control emerged under the general heading of situated reasoning. These approaches stress a tight coupling of sensing and action to deal with changing situations and data uncertainty. Two such approaches are reaction plans [Schoppers, 1989] and subsumption [Brooks, 1986]. Essentially an enumeration of actions to take for all possible preconditions, reaction plans have a strong appeal for those interested in field robots. The ability to retry a subtask in the face of a changing situation, detected through continuous sampling of the environment, promises robust execution of tasks. Additionally, such plans can be generated using formal techniques, providing a theoretical basis for plan analysis. Yet, for any useful application, it has been shown [Ginsberg, 1989] that such plans, if required to map raw sensor input into primitive actuator output, can quickly become exponentially large.

The subsumption approach also has a strong appeal. This is not only because it has been used successfully on actual robots, but also because as higher level competences are developed, the competences of the lower layers of the architecture are retained as default behaviors for unexpected or critical situations. Yet the subsumption approach has generally not included room for goal representations to go beyond what is sometimes termed insect intelligence.

This paper describes an intelligent agent architecture wherein reaction plans use subsumption competences as operators. The key benefits are that the reaction plans need only map agent states into competences, thus reducing their size, and that the formal goal representation of the reaction plan can augment the insect intelligence of the subsumption competences. In our work we commit to layered control and to the combining of directives between layers. But we do not commit to the original subsumption language, asynchrony of finite state machines, or the use of inhibition and suppression techniques. Thus, the term layered competences is used rather than subsumption for the remainder of this paper.

In considering implementing the architecture, we wanted the resulting code to run synchronously. One of the thrusts of our work is to analyze agent operations for safety [Bonasso *et al.*, 1990]. If the executing software has guaranteed constant cycle times from sensor input to actuators commands, we can isolate the software operations from sensor and actuator physics for analyzing ranges of safe operations. Constant time software requires synchronous operations. We also wanted to insure that the formal semantics of any abstract representations still held in the on-board software. Synchronous operations allows for simpler semantics about the agent's most recent perception of the world and the formal rationale for carrying out the next action.

The desire for synchronous operations and consistent semantics led us to implement the architecture in the GAPPS/Rex language [Kaelbling, 1988]. The language brings to bear formal semantics about the relationship between an agent's internal states and those of the

environment [Rosenschein and Kaelbling, 1986]. As well, the accompanying robot programming environment can build both the reaction plans and the layered competences of the architecture, while the resulting circuits guarantee constant cycle times.

## 2 Domains of Interest

We are initially interested in robots carrying out retrieval, delivery and reconnaissance tasks in field environments. These tasks consist of general navigation from the start point to the area of interest, grasping and ungrasping or directed sensing, and then general navigation back to the start point or subsequent sites. Though not complex, these tasks are the basis for many jobs robots are expected to carry out in field environments. Loading and unloading cargo, explosive ordinance disposal, planetary exploration, aerial tracking of ground phenomena, deep ocean mapping and exploration, all fundamentally consist of retrieval, delivery or reconnaissance tasks.

Because the architecture does not yet address plan development at runtime (but see Other Related Work), we further assume that the robot can be informed by other agents or humans as to changes in its mission.

We have begun experiments with several robots for such tasks as described above, and current results show that the architecture works well under these conditions.

## 3 Proposed Architecture

As stated above, the architecture consists of reaction plans which use competences for operators. Figure 1 shows a sample reaction plan for a delivery task for a one-armed mobile robot in our indoor experiments. The preconditions of the plan are generated by hand for simple plans, or automatically via goal regression for more complex plans. On each pass of the control loop, these preconditions, shown here as a logic vector, are tested and the appropriate action is taken.

The numbers to the left show the linear order of subtasks to accomplish the delivery task. Task activity commences when a widget appears, at which time the robot obtains the widget to be delivered and locates the receiver. The robot then moves to the receiver and places the widget on the receiver platform. Finally, the robot backs away from the receiver and retracts the arm. Since the preconditions are checked continuously, the reaction plan provides for robust operations. For example, if the receiver moves during the MOVE-TO operation (step 5), the receiver-located precondition will be false, causing the robot to re-acquire the receiver (step 4). Thus, the robot will track the receiver while moving toward it. Likewise, if while the robot is raising its arm (step 6), the platform moves under the grasped widget, then the widget will be over the platform and the robot will skip step 7 and execute the PLACE operation of step 8.

This plan is sufficient at the given level of detail for delivery tasks with actual robots in field environments only if the operators such as MOVE-TO, GET-CLEAR and GRASP are competent. That is, the operators must be able to deal with variations in the environment as part of their design, thus unburdening the reaction plan of those considerations. This is the key contribution of this work to the growing body of situated reasoning research. Figure 2 shows the concept for the layered competences in the architecture. The figure is notional. For instance, the lowest level perception results could be made available to the highest level reasoning. And there may be more than three layers of competence, though in practice, we have used the three layers shown.

There are three hallmarks of these layered competences. The first is the basic trait of the subsumption architecture: higher level competences subsume those of the lower levels. For example, once the robot has a competence to avoid obstacles on the fly, any high-level navigation vector which is generated by, say, a MOVE-TO reaction plan operator, will be adjusted to insure avoiding a previously unseen obstacle. This merging of directives (shown by the valve icon in Figure 2) can take the form of simple constraints placed on the higher layers by the lower layers or can involve more sophisticated combination algorithms. In our work with an underwater robot, it was sufficient to average the 3D obstacle avoidance vector with the navigation mission vector. In our work with a ground mobile robot we use a combination algorithm based on a template of geometric constraints [Slack, 1990] for robust outdoor navigation.

The second hallmark is that the lowest level dictates the smallest cycle time, and higher-level cycles are multiples of that time. Our implementation generates synchronous circuits which, at each strobing or tick of the circuit, guarantees outputs for the lowest-level competence. Subsequent ticks may produce additional outputs from higher levels in the architecture. This insures that the lower levels can be configured to effect emergency reactions tailored to the fastest problematic events in the environment, and yet those reactions will be blended with higher-level outputs as they become available.

The third hallmark is that levels of perception processing roughly match the levels of reactive competence, i.e., that perception at each level is task-driven. Thus, in the implementation, there may be global structures to allow for search efficiency, but task-related perception algorithms, if not individual representations, can exist at each level. For example, in one of our developments, the proposition (aware-p ?class-of-things) is used as part of the LOCATE competence (top layer). If the predicate is not true, then the robot's database of objects must be updated via a directed sensor search algorithm. At the lowest layer, for obstacles, the agent becomes aware simply by receiving raw data, such as the signal from a bumper contact switch.

## 4 Implementation

As mentioned before, we have chosen to use the GAPPS/Rex robot programming environment for our implementation, since the resulting circuits are constant time and have formal semantics. The architecture dictates a programming methodology as follows. First write GAPPS goal reduction rules for the invocation of competences. With the rules in Figure 3, for example, a HERO robot will turn

| | Preconditions | | | | | | | | Actions | Comments |
|---|---|---|---|---|---|---|---|---|---|---|
| | WIA | ERA | WIG | RCL | DTR | AIP | WOP | WIP | | |
| 10 | * | T | F | * | * | * | * | T | RETIRE arm. set WIA toF | Task complete |
| 9 | * | F | F | T | T | T | * | T | GET-CLEAR of receiver | So arm can be retracted |
| 8 | * | * | T | T | T | T | T | F | PLACE widget | A simple release |
| 7 | * | * | T | T | T | T | F | F | GET-NEAR receiver | So widget wiN be over platform |
| 6 | * | * | T | T | T | F | F | F | RAISE arm | So arm is at platform level |
| 5 | * | * | T | T | F | * | * | F | MOVE-TO receiver | Arm Is oriented in front of robot |
| 4 | * | * | T | F | * | * | * | F | LOCATE receiver | Via perception or being informed |
| 3 | T | T | F | * | * | * | * | F | GRASP widget | After getting some elbow room |
| 2 | T | F | F | * | * | * | * | F | POSITION-FOR-GRASP | Or accept from a donor |
| 1 | F | * | * | * | * | * | * | * | Stay out of trouble | Default; no plan; survive, energy conservation.etc. |

T= true, F=false, * = don't care
WIA = Widget Is Available, ERA = Ebow Room Available
WIG = Widget In Grasp, RCL = Receiver Located
DTR = Distance To Receiver < reach distance
AIP = Arm in place position, WOP = Widget over platform
WIP = Widget Is Placed

Figure 1. A Reaction Plan for Widget Delivery

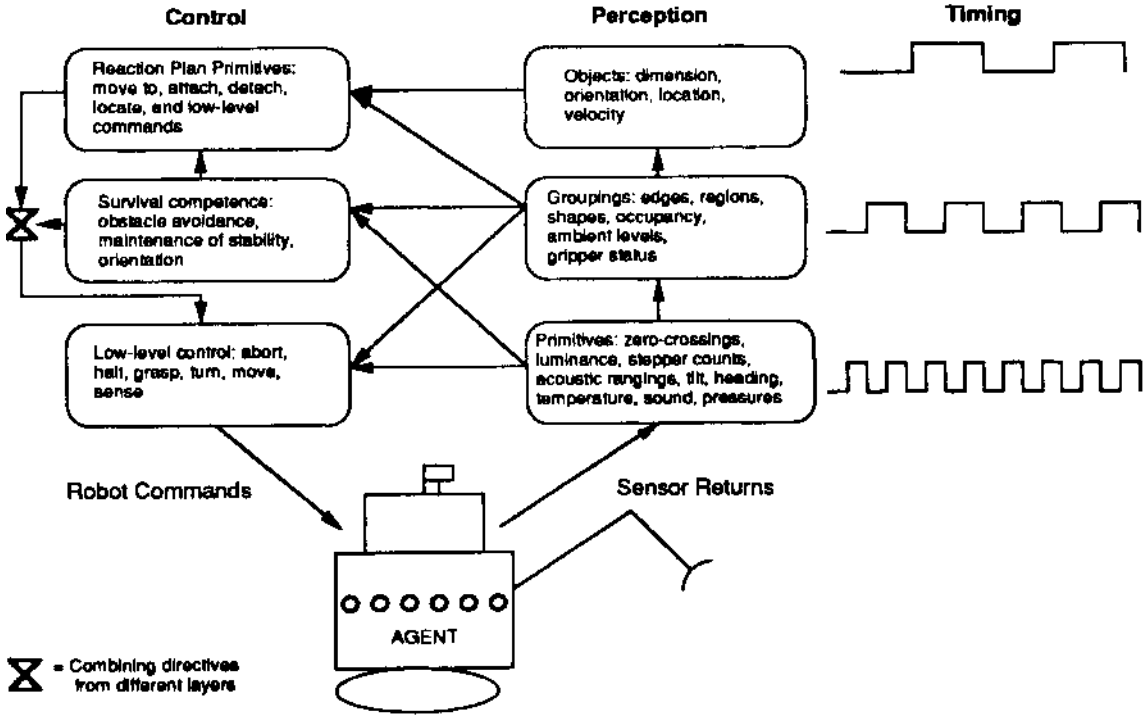


Figure 2. Layered Competences

from the nearest obstacle and move slowly away. Figure 4 shows similar rules for a wander operation.

After writing these rules, the next step is to write the Rex code for the functions that make up the competence execution, such as the (set-avoidance-turn <arg>) function and the (moving-slowly) and (no-obstacles) predicates in the example.

Finally, one builds the reaction plan using the goal expressions in the reduction rules. A simple plan can be built using a prioritization of goals. The GAPPS (prio-and gl ... gn) tries to satisfy n goals, but failing that tries to satisfy n-1 goals, etc. An example plan to have a HERO robot wander safely around an area would be:

```
(prio-and (maint not-crashed)(ach wander))


(defgoalr   (maint not-crashed)
            (if     (no-obstacles)
                    (do anything)
                    (ach avoid nearest obstacle)))

(defgoalr   (ach avoid nearest obstacle)
            (if     (notm (Hero-at-avoidance-angle))
                    (ach turn to avoidance angle)
                    (if     (notm (moving-slowly))
                            (ach moving slowly)
                            (do anything))))

(defgoalr   (ach turn to avoidance angle)
            (and    (do update? !*turn-command*)
                    (do left-motor-dist
                       (set-avoidance-turn !*left*))
                    (do left-motor-speed
                          !*caution-drivc-speed*)
                    (do right-motor-dist
                       (set-avoidance-turn !*right*))
                    (do right-motor-speed
                       !* caution-drive-speed*)))

(defgoalr   (ach moving slowly)
            (and    (do update? !*move-command*)
                    (do left-motor-dist !*max-move-dist*)
                    (do left-motor-speed !*caution-drive-speed*)
                    (do right-motor-dist !*max-move-dist*)
                    (do right-motor-speed
                       !*caution-drive-speed*)))
```

Figure 3 . GAPPS reduction rules for obstacle avoidance. Terms with asterisks are global parameters which are formed into a structured memory location by the ! symbol. Ach and maint are abbreviations for achieve and maintain respectively. Notm is a Rex machine for the not function. The do command essentially sends the specified value to a vector of outputs.

An abstract of the resulting circuit is shown in Figure 5, where it is compared to a wander circuit from the subsumption approach. The two circuits are surprisingly similar: they both have wander and runaway competences, make use of a sonar map, and receive feedback via optical encoders on the robot's wheels. Moreover, the performance of the GAPPS wander circuit, implemented on our Denning mobile robot, is the same as that of the robot using the subsumption architecture; i.e., the robot can wander aimlessly about for hours without colliding with stationary objects or slow-moving humans.

```
(defgoalr   (ach wander)
            (if     (notm (Hero-at-wander-angle))
                    (ach turn to wander angle)
                    (if     (notm (moving-at-speed))
                            (ach moving-at-speed)
                            (do anything))))

(defgoalr   (ach turn to wander angle)
            (and    (do update? !*  turn-command*)
                       (do left-motor-dist (set-wandcr-tum !*left*))
                       (do left-motor-speed !* normal-drive-speed*)
                       (do right-motor-dist
                                   (set-wandcr-turn !*right*))
                       (do right-motor-speed
                                      ! *normal-drive-speed*)))
```

Figure 4. GAPPS rules for wandering. The function set-wander-turn uses the sonar readings to find open spaces.

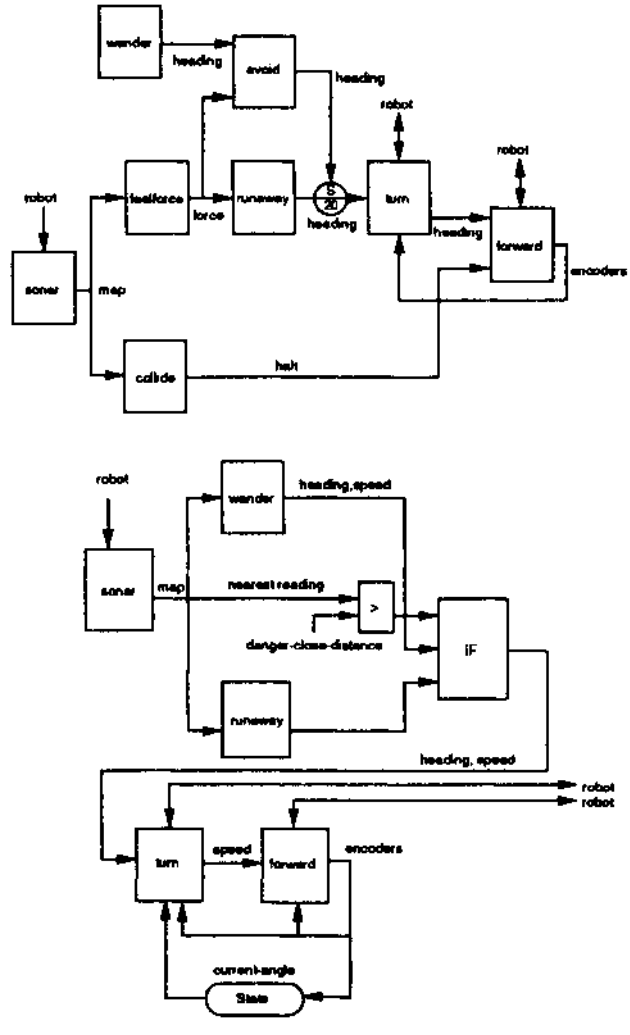


Figure 5. Circuits for wander routines from [Brooks, 1986] (above) and layered competences from GAPPS goals (below)

There are important differences. The avoid block and the IF block both serve to produce the correct heading under the right conditions, but the avoid block is part of the wander competence, whereas the IF block specifically mediates *between* competence layers as a result of specifying a natural prioritization of high-level goals. Also, the GAPPS circuit is a synchronous circuit. That is, it reads the sensors and computes outputs to the wheels in continuous cycles of a guaranteed constant time. The subsumption circuit functions as a network of asynchronous finite state machines whose timing is a function of the environmental conditions. Finally, the GAPPS circuit allows for states, such as the robot's current heading angle, used in the (HERO-at-wander-angle) and (HERO-at-avoidance-angle) predicates. Because of the circuit synchrony, it can be shown that the robot knows in a formal sense its current angle with respect to its actual position in the environment [Rosenschein and Kaelbling, 1986].

With the architecture described in this paper, we can make the high-level goal specification even more complex as in the hand-crafted reaction plan shown in Figure 6. A recent extension to GAPPS described in [Kaelbling, 1990] supports goal regression and allows the GAPPS compiler to generate these plans automatically. It is not clear that this can be done in the original subsumption architecture or programming language.

A further advantage to using GAPPS is that we can organize the merging of constraints and the combination of competences differently for different tasks at compilation time. For example, the GAPPS coding in Figure 7 invokes an obstacle avoidance competence via the (maint not-crashed) goal during a navigation action. Figure 8 shows how a simple change to the top-level goal makes the obstacle avoidance competence active during the entire mission. This kind of compile-time wiring can carry down

```
(defgoalr   (ach place object on platform)
   (if (new-mission-in- inputs)
      (if (andm (notm (object-placed)) (notm (have-object)))
         (ach get-object)
         (if (andm  (notm (object-placed))
                    (notm (arm-in-raised-position)))
            (ach arm-raised)
            (if (andm  (notm (object-placed))
                       (notm (near-enough-to-platform)))
                       (ach get near to platform)
                       (if (notm (object-placed))
                          (ach place-object)
                          (if (andm (object-placed)
                                    (notm
                                       (HERO-clear-of-platform)))
                             (ach HERO clear of platform)
                             (if    (andm   (object-placed)
                                            (notm (HERO-retired)))
                                    (achretire-HERO)
                                    (do anything)))))))
      (do anything)))
```

Figure 6. A reaction plan for object delivery by a HERO 2000 to a mobile platform.

```
;; This reduction rule says if there is no new-mission input by
;; the user, then look for an object to be delivered. Else place the
;; object on the receiver platform.

(defgoalr   (ach carry out any missions)
            (if    (notm (new-rnission-in-inputs))
                   (ach look for new mission)
                   (ach object placed on receiver platform)))

;; To "look" for a new mission, look for an object in the
;; strongest light (robot has light and proximity sensors, no
;; camera). Else try  to move to the strongest light while not
;; bumping into anything. If there is an object in the light, the
;; robot sets a possible-new-mission flag in the output which
;; will  trigger an input routine to ask the user if there is indeed a
;; mission with the object in question.

(defgoalr   (ach  look-for-new-mission)
            (if    (some-object-exists-in-the-light)
                   (and     (ach turn to object in the light)
                            (do possible-new-mission? !yes))
                   (prio-and (maint not-crashed)
                             (ach move-toward-strongest-light))))
```

Figure 7. Reduction rules for (ach carry out any missions)

```
(defgoalr   (ach carry out any missions)
            (if    (notm (new-mission-in-inputs))
                   (ach look for new mission)
                   (ach object placed on receiver platform)))

;; To "look" for a new mission, look for an object in the
;; strongest light (robot has light and proximity sensors, no
;; camera). Else try to move to the strongest light .

(defgoalr   (ach  look-for-new-mission)
            (if    (some-object-exists-in-the-light)
                   (and     (ach turn to object in the light)
                            (do possible-new-mission? !yes))
                            (ach move-toward-strongest-light)))
```

Figure 8. Reduction rules for the goal
(prio-and (maint not-crashed) (ach carry out any missions))

through the competences via the Rex functions, though the results are not as perspicuous in the Rex code as in the GAPPS language.

The original GAPPS/Rex environment generated a single circuit for each compilation. Code could be arranged in groups of modules which represented the layers of competences, but all modules were constrained to execute to completion in one cycle, making the cycle time proportional to the size of the enure circuit If sufficient CPU speed or a parallel architecture is readily available, this is not a problem. However, most commercially available platforms have the equivalent of an M68000 or less, so we needed to adhere to the architecture's proportional timing relationships among levels to insure that the most critical "reflex" actions could execute in a minimum time cycle. A recent change to the Rex compiler allows us to code competences in accordance with the architecture's timing relationships, by

specifying a maximum execution time of submodules of code.

## 5 Other Related Work

The work by Chapman and Agrc [1987] can be grouped with subsumption as research in intelligence emerging strictly from activity. Our layered competences generate behaviors exhibiting a certain amount of emergent intelligence. However, with reaction plans, we add goal representation and explicit ordering of behavior priorities.

A direct alternative to the reaction plan is Firby's reaction action packages (RAPs) [1989]. The RAPs depend on an extant primitive execution layer, which could be the layered competences of the above architecture. The semantics of the RAPs are formally derived, but it is not clear how they extend to the machine level.

The research in subsumption, situated automata, and emergent intelligence (e.g., [Chapman, 1990]) all use circuit languages. Another circuit language is Gat's Behavior Description Language (BDL)[Gat and Miller, 1990). This makes first class objects out of the channels that connect behaviors. Like the original Rex system, BDL results in a synchronous system where all behaviors execute on each cycle, and, like GAPPS, making the connections explicit allows the programmer to arrange the behaviors to suit the task (and/or hardware) available. No formal semantics of the behavior states have been worked out, but analysis might show them to be similar to those of Rex.

Recent work by Maes [1990] is similar to the work described here in that she uses competences for plan operators as we do in the reaction plans. It is dissimilar in that she uses explicit pre/post conditions for sequencing competing behaviors rather than subsumption. Also her goal is run-time planning which the architecture in this paper does not yet address. [Kaelbling, 1990] discusses how runtime planning might be done in GAPPS, and we are also looking at integrating deliberative and situated paradigms (see for instance [Elsaesser and Sanborn, 1990]).

## 6 Experiments

In our current experiments, we are using two HERO 2000 mobile robots with 5-degrees of freedom arms, a Denning MRV-III mobile robot, and a remotely-piloted vehicle (RPV) for undersea operations at the Woods Hole Oceanographic Institute (WHOI). As of this writing our results are qualitative, i.e., the robots carry out their tasks as expected without getting into trouble in unknown or partially known environments. We are currently attempting to quantify the results with measures of effectiveness such as number of collisions or the task completion time as functions of the number and arrangement of environmental obstacles.

The architecture was originally developed in the context of a simulation of cooperating agents executing retrieval tasks, and was subsequently implemented in GAPPS/Rex on a HERO to execute the wander behavior. The HERO uses ultrasonic sensors, optical encoders on its wheels, arm joints, and gripper, and a light intensity sensor. A code generator was developed to translate the Rex circuits into the HERO'S native BASIC language for downloading and subsequent on-board execution. Presently, the HERO executes a delivery task in an indoor laboratory environment (see Figures 6 and 7). A mission is signaled to the robot by a human's proximity. The robot distinguishes the human from other objects by asking questions. Then the robot accepts the object from the human and moves to a mobile platform where the object is deposited. A flashlight is used to identify the receiver platform (the room lights are dimmed during the runs). The HERO avoids obstacles and slow-moving other agents, and deals robustly with the receiver platform being moved mischievously by humans under joystick control during the PLACE operation.

We have implemented layered competences in GAPPS/Rex on the Denning MRV-III mobile robot for navigation among indoor and outdoor obstacles. The Denning has a ring of 24 acoustic sensors, an infra-red navigation beacon system, and an inclinometer. The Denning hosts an M68000 running the OS9 real-time operating system. The GAPPS runtime environment for Unix was adapted for OS9, so that complete GAPPS circuits generated in C can be run autonomously on the Denning. However, our circuits using more than simple vector techniques are too large to achieve useful responses on-board, so we typically use an RS232 tether to a faster computer.

The robot accurately achieves and maintains a 20-50 foot trajectory in the face of both natural and human-introduced obstacles in our Autonomous Systems Laboratory environment and in one of our employee parking lots. The robot will halt and adjust its heading when a human passes by unexpectedly. Currently, to avoid automobiles moving faster than its one foot per second speed, it relies on the human intelligence behind the wheel. The robot will soon be instrumented with a real-time stereo system [Bonasso and Nishihara, 1990] to conduct outdoor retrieval tasks.

The WHOI RPV has a transponder-based navigation system for location updates, and an adaptive trajectory control algorithm which we use as our MOVE and TURN competences (see [Bonasso, 1991]). The navigation computation and control algorithms execute on a transputer architecture. To improve the vehicle's awareness, a three-dimensional Proximity Obstacle Detection System (PODS) using sonar altimeters was designed and integrated on board the vehicle. We then used the architecture to successfully develop a pilot support system wherein a human pilot steers the vehicle while the vehicle autonomously avoids obstacles and the walls of a test tank. Thus, the architecture was able to integrate the natural intelligence of the pilot (the reaction planner), the heuristic obstacle avoidance behavior, and the analytical competence of the control-theory routines. Follow-on experiments, based on these results, are being planned for a second vehicle with a manipulator, and for a deep ocean mapping task.

It has been our experience that once one is familiar with the GAPPS/Rex system, useful working robot programs can be constructed literally overnight. We used the same basic programming methodology outlined in this paper for different tasks, with different robots, both for semi-autonomous and autonomous operations. All of these codes were developed via the GAPPS language while adhering to

die principles of the above described architecture essentially in a few days, resulting in the robots being endowed with task-achieving capability almost immediately. Thus, there are indications that the architecture and its implementation promise to be generally useful for a variety of tasks on a variety of robots. However, the tasks in the above experiments have not required extensive sensor processing. The outdoor retrieval task with stereo vision, and the ocean mapping tasks use more sophisticated sensors and should do much to confirm or deny these preliminary indications.

## 7 Conclusions

We have developed an architecture for retrieval, delivery and reconnaissance tasks which integrates the intuitively attractive traits of reaction plans and subsumption for the control of robots in field environments. The approach adds to each what was lacking — robust operators for the reaction plans and a goal representation for subsumption. We attain desired synchronous operations and formal semantics by implementing the architecture in GAPPS/Rex. This programming environment allows us to build flexible combinations of competing behaviors as appropriate for the task.

So far, experiments seem to qualitatively bear out the utility of our approach. The robots used perform their tasks robustly during normal changes in the task environment. In addition, analytical competences from other disciplines or natural intelligence have been easily integrated into this architecture with good effect. Further experiments with additional sensors in more complex environments are underway.

## Acknowledgements

We wish to acknowledge the excellent efforts by Leslie Kaelbling and Nathan Wilson in making the necessary changes to GAPPS/Rex which support the architecture described in this paper. Thanks are due also to Jim Antonisse, Lashon Booker, Leslie Kaelbling, Stan Rosenschein, and Marc Slack for helpful comments on an earlier draft of the text.

## References

[Bonasso and Nishihara, 1990] R. Peter Bonasso and H. Keith Nishihara. Using Real-Time Stereopsis For Mobile Robot Control. In *Proceedings of the International Symposium on Applications in Optical Science and Engineering,* Vol 1387, pages 237-244, Boston, July 1990. SPIE.

[Bonasso *et al.,* 1990] R. Peter Bonasso, Vincent Hwang, James Sanborn, and William Stoney. Investigating Robot Safety and Robustness In An Autonomous Systems Laboratory. In *Proceedings of the 1990 International Conference on Space Applications of AI, Robotics, and Automation,* pages 105-108, Kobe, Japan, Nov. 1990. AIAA.

[Bonasso, 1991] R. Peter Bonasso. Underwater Experiments With A Reactive System For Autonomous Vehicles. In *Proceedings of the 9th National Conference on Artificial Intelligence.* Anaheim, CA., July 1991. AAAI Press.

[Brooks, 1986] Rodney A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation,* RA-2:14-23, April 1986.

[Chapman, 1990] David Chapman. *Vision, Instruction, and Action.* PHD Dissertation. MIT TR # 1204. 1990.

[Chapman and Agre, 1987] David Chapman and Philip E. Agre. Abstract Reasoning As Emergent From Concrete Activity. In *Reasoning About Actions and Plans,* pages 411-424. Los Altos, CA, 1987. Morgan Kaufman.

[Elsaesser and Sanborn, 1990] Christopher Elsaesser and James Sanborn. An Architecture for Adversarial Planning, *Systems, Man, and Cybernetics,* 20(1): 186-194, Jan-Feb, 1990. IEEE.

[Firby, 1989] James R. Firby. *Adaptive Execution in Complex Dynamic Worlds.* PHD Diss. YALEU/CSD/RR #672, Yale University. 1986.

[Gat and Miller, 1990] Erann Gat and David Miller. *BDL: A Language For Programming Reactive Robotic Control,* JPL Pub, forthcoming.

(Ginsberg, 1989] Matthew L. Ginsberg. Universal Planning: An (Almost) Universally Bad Idea. *AI Magazine,* 10(4): 40-44,1989.

[Kaelbling, 1988] Leslie Pack Kaelbling. Goals As Parallel Program Specifications. In *Proceedings of the Seventh National Conference on Artificial Intelligence,* pages 60-65, Minneapolis-St. Paul, Minnesota, 1988. AAAI Press.

[Kaelbling, 1990] Leslie Pack Kaelbling. *Compiling Operator Descriptions into Reactive Strategies Using Goal Regression.* TR90-10. Teleos Research. Palo Alto, CA. 1990.

[Maes, 1990] Pattie Maes. Situated Agents Can Have Goals. *Robotics and Autonomous Systems,* 6(1&2): 49-70, 1990.

[Rosenschein and Kaelbling, 1986] Stanley J. Rosenschein and Leslie Pack Kaelbling. The Synthesis of Digital Machines with Provable Epistemic Properties. In *Proceedings of the Conference on Theoretical Aspects of Reasoning About Knowledge,* pages 83-98. Morgan Kaufman. 1986

(Schoppers, 1989] Marcel J. Schoppers. In Defense of Reaction Plans As Caches. *AI Magazine,* 10(4): 51-60, 1989

[Slack, 1990] Marc G. Slack. *Situationally Driven Local Navigation for Mobile Robots,* JPL Pub. 90-17, 1990. NASA.