N90-27303

CREATURE CO-OP : Achieving Robust Remote Operations With A Community of Low-Cost Robots

R. Peter Bonasso (bonasso@ai.mitre.org)
Washington Artificial Intelligence Technical Center
The MITRE Corporation
7525 Colshire Drive, McLean, VA 22102

Abstract

This paper puts forth the concept of carrying out space-based remote missions using a cooperative of low-cost robot specialists rather than monolithic, multi-purpose systems. A simulation is described wherein a control architecture for such a system of specialists is being investigated. Early results show such co-ops to be robust in the face of unforeseen circumstances. Descriptions of the platforms and sensors modeled and the beacon and retriever creatures that make up the co-op are included.

INTRODUCTION

An alternative to building one robot for remote planetary operations, like the Mars Rover, is to build several robots whose combined capabilities can do the same task. A single Mars Rover will have extensive perception capabilities, robust navigation capabilities, intricate sample detection and acquisition capabilities, long-range fuel capability, and, in some designs, an orbit-achieving propulsion system. But damage to any major subsystem will halt the reconnaissance mission for months until another Rover can be launched and landed on Mars.

An alternative approach is to use a cooperative of robots of several sorts. One sort can carry fuel cells (fuel-bots), another sort has strong farsighted visual perception (beacon-bots), another sort has good mobility and dexterity and short-sighted vision (rover-bot), another sort can attach itself to objects and push and pull them (retriever-bot), and so forth. Each robot would have the basic capability to stay out of harm's way (e.g., avoiding sand pits, dodging meteors), and could communicate with the others via RF or some other wireless link.

Then, in the Mars Rover effort for example, the beacon-bots and fuel-bots could be placed in strategic positions throughout the reconnaissance area, serving as far-sighted eyes and gas-stations for the rovers which, near-sighted, negotiate rough terrain and acquire soil samples. The earth-bound human controller will always be in the loop, evaluating the evolving situation and sending high-level commands (but only every 40 minutes or so). Losing one of the beacon-bots or fuel-bots will only limit the overall system capability in a certain geographic area. Losing a rover-bot or retriever-bot will only burden another rover or retriever with more tasks to do of the same type. So this

community of robots could be ultimately more cost-effective than one large all-purpose, well-equipped robot.

This paper describes an ongoing investigation into robot cooperatives for accomplishing remote operations. It builds on research being carried out in situated reasoning, perception, and planning for autonomous agents. Specifically, this work relies on the ideas of reasoning via subsumption software architectures (3, 6) and reaction plans inspired by (13), and on the growing number of low-cost integrated robot platforms that are becoming available (e.g., (4), (5)). The idea of distributed robots is not new, dating at least to early-eighties work by Sacerdoti who coined the term disbots (12). But only in recent years has realizing such systems on low-cost integrated platforms been possible.

This paper does not address many difficult issues associated with distributed control. The assumption here is that a low-level of cooperation can be achieved much like a *hive* gestalt, and anything beyond that is handled by human intervention.

The next section details an object-oriented software environment on a micro-Explorer for conducting experiments in robot cooperation. These experiments are beginning to demonstrate that such cooperatives will be robust and relatively inexpensive. Through an example, the cooperating capabilities of the "creatures" (see (3)) are characterized and the implications for cost discussed. The last section describes follow-on work with actual robots in MITRE's Autonomous Systems Laboratory.

THE CREATURE CO-OP EXPERIMENTAL ENVIRONMENT

Figure 1 shows a top or plan view of the layout of the environment on a micro-Explorer computer screen to simulate the action of a cooperative of robots with varying levels of competence. The large circle is the work area within which all robot action takes place. The open geometric figures are candidate objects for retrieval. The filled-in circles are the creatures for the described experiments: the smallest ones are beacons with farsighted vision; the others are retrievers with circumferential sonars, and arms (initially retracted) that extend from the body to grasp or latch on to the object to be retrieved. The thin lines extending from each creature show their current orientation (they are all at 0 degrees; angles are measured counter-clockwise).

The right-hand pane shows a number of menu options available to the experimenter and the bottom pane is used for text output and user typein. The user can direct the creatures to retrieve objects, move to new locations, go to sleep and wake up.

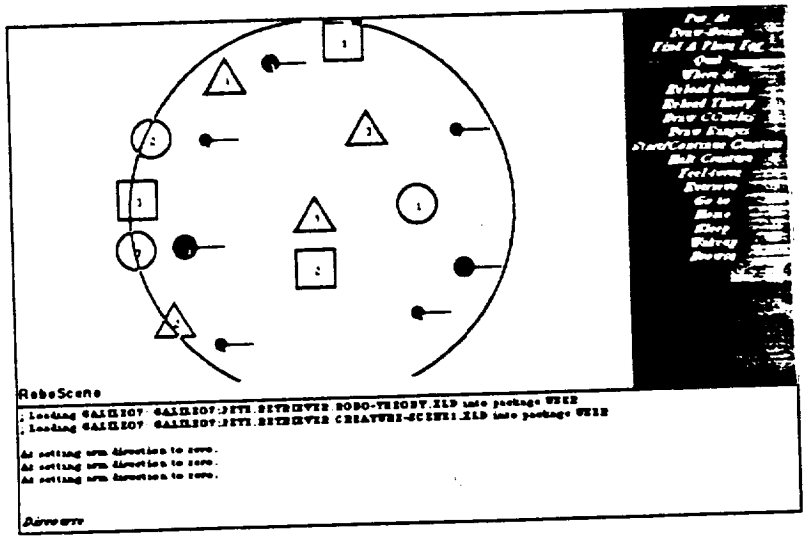

Figure 1

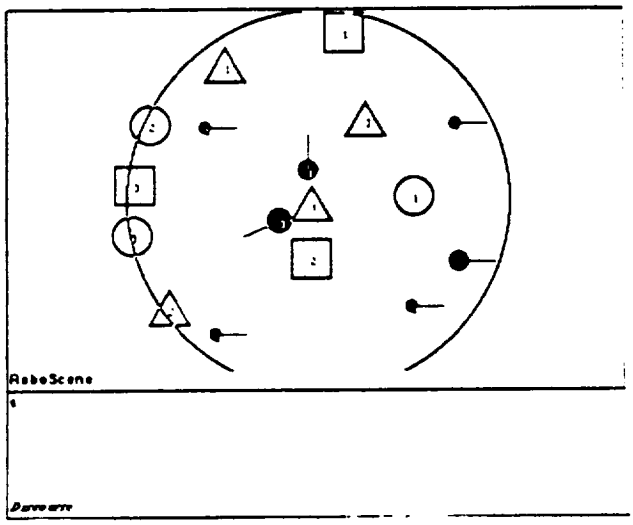

Figure 2

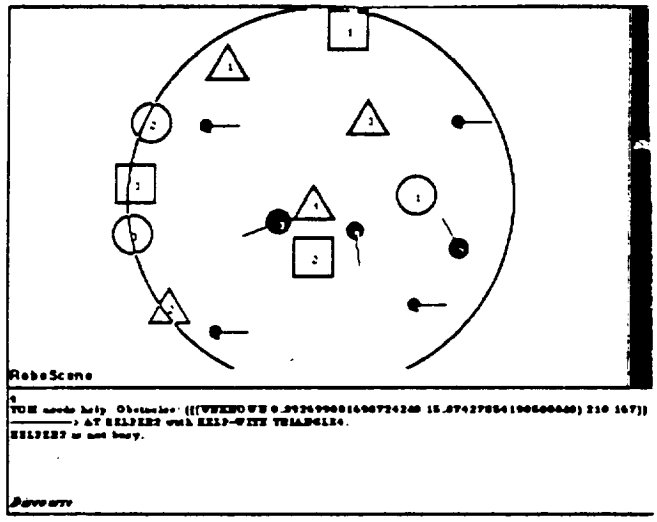

Figure 3

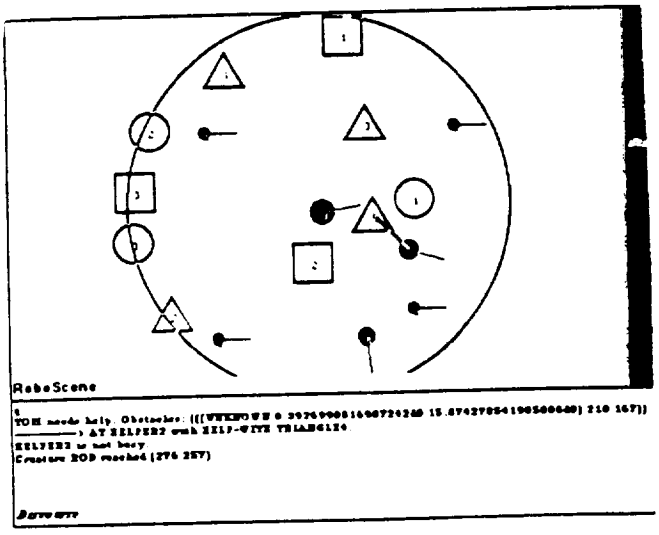

Figure 4

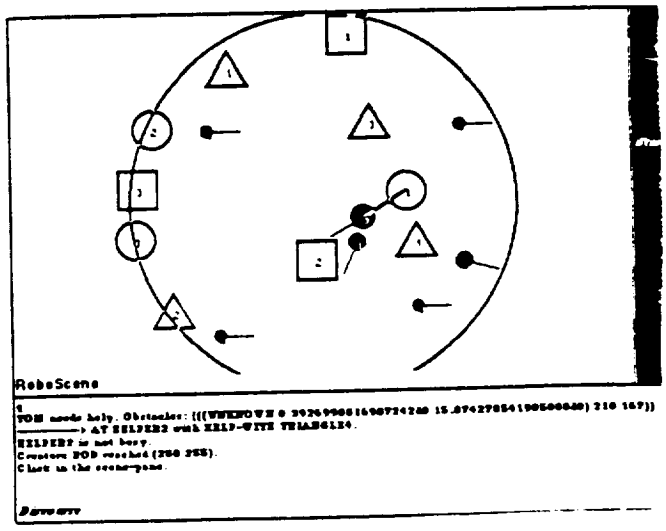

Figure 5

An Example Of Creature Cooperation

The following example demonstrates the utility that can be obtained with minimal competency "specialist" robots in a cooperative effort. In this example, the large retriever (#3) has the task to retrieve Cylinder1, and the small retriever at the top of the screen (#1) is to navigate to coordinates in the bottom part of the work area and return.

Selecting Start/Continue Creature generates a process for each creature that for each simulation clock tick causes the creature to continue to carry out a retrieval or a directed move and to avoid any obstacles . Without any retrieval or directed moves pending, no action would be detected. But if the user puts an object next to any creature, that creature would immediately move a safe distance away.

After being given a target-object, retriever #3, blind but for its short-range sonars, first sends a request via RF link for the location of Cylinder1. Only beacons have the requisite ranging and recognition capabilities. The beacon in the upper right of the picture locates Cylinder1 and transmits the coordinates to retriever #3, which begins to dead reckon to the object. Retriever #1 simply heads for the desired location at the bottom of the screen.

In Figure 2 both retrievers have encountered Triangle4. Both creatures have a low-level survival routine that usurps control from any higher level routine when an obstacle is sensed. The "angle of escape" and "speed of escape" computed by this routine are averaged with the "angle of purpose" and "speed of purpose" of any higher level routines to allow the retriever to angle back on track once the obstacle has been avoided. In Figure 3, retriever #1 is past the obstacles and is heading unobstructed for its destination.

Retriever #3, however, is blocked since in attempting to reach Cylinder1 it is ping-ponging between the Triangle4 and Block2. At this point, noticing that it is not making progress (see Getting Stuck below), retriever #3 broadcasts a help request indicating an obstacle near the coordinates of Triangle4. Any retriever that is not busy with another task can process the help request, and retriever #2 (to the right of Cylinder1) takes up the task, broadcasting a request for a beacon to identify the object near the coordinates of interest. Again, the beacon in the upper right locates and identifies Triangle4 for retriever #2.

When retriever #2 is close enough to the triangle, it extends its arm, latches onto the object, and begins the return trip to the location from which it received the help request, thus pulling the obstacle out of the way of retriever #3. Figure 4 shows retriever #2 returning and retriever #3 continuing toward Cylinder1. Retriever #1 has arrived at the required destination.

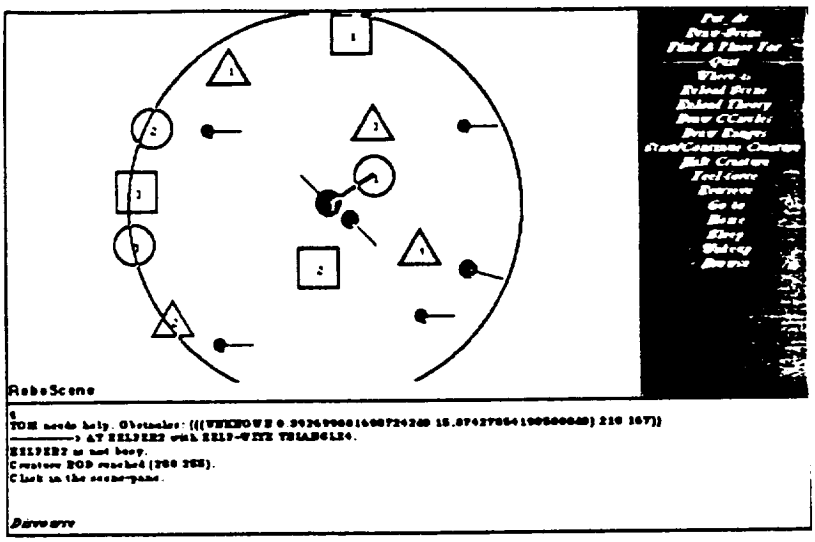

Figure 6

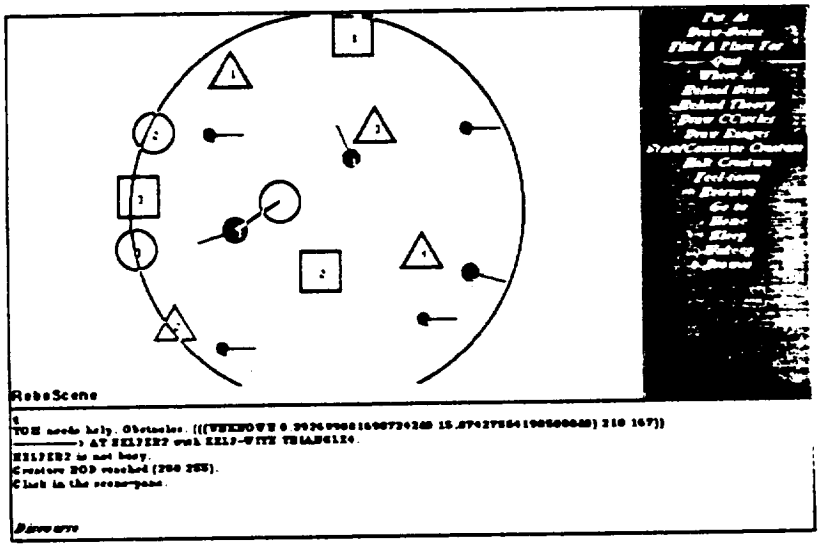

Figure 7

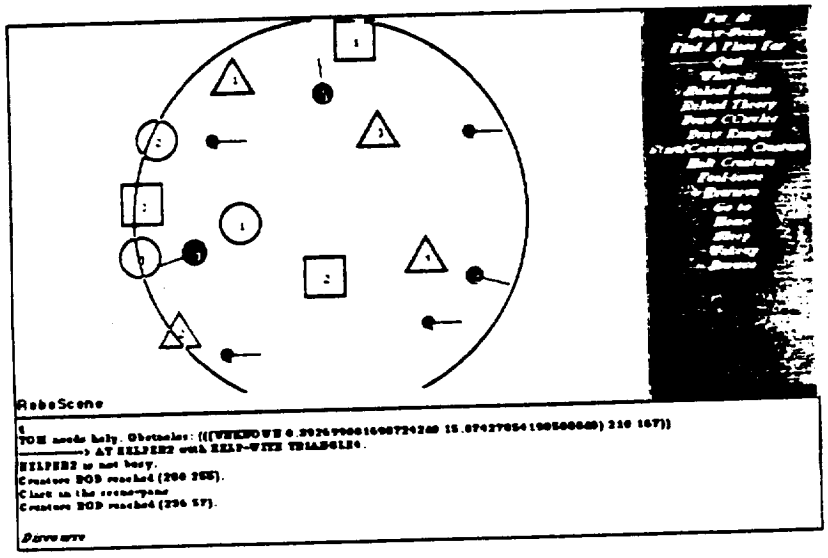

Figure 8

Figure 5 shows retriever #3, which is returning to its point of origin with Cylinder1, encountering retriever #1, which is returning to its point of origin. Both are blind but can avoid obstacles with their sonars, so they jostle each other as they move toward their destinations (see Figure 6). In Figure 7, both creatures are again on their way to fulfilling their tasks which are completed in Figure 8. The human user has simply specified two high-level tasks, and the retrievers were able to achieve them despite obstacles and the crossing of paths during the task execution.

CO-OP Objects and Operations

Table 1 shows the objects and operations used to achieve the capabilities discussed in the text of this report. There are six objects: three sensors, the generic one and the proximity and vision types; and three creatures, the generic one and one with and one without an arm. The types inherit data and operations from the generic objects. Operations are prefixed by colons; operations that are invoked after another operation are written with the word :after and the precedent operation name.

Each creature is instantiated via its position and orientation variables and the type and number of sensors it is given.

Object	Type	Variables	Operations
sensor		associated-creature, type	
	proximity-sensor	range-factor	:sense-from coords
	vision-sensor	range	:locate sought-object, :identify-object-at
creature		shape, number, cc-radius, coords, sensor-list, name, sleep	:sleep, :wakeup
	round-creature	speed, direction, target-direction, old-coords, caution-speed, obstacle-memory	:feel-force,:sense, :runaway,:move, :turn new-direction, :locate object, :locate-object-at coords, :wander
	round-creature-with-arm	target-object, target-location, arm-length, arm-direction, target-object-grasped, arm-extended?, blocking-obstacles, help-is-on-the-way, creature-being-helped, distance-and-time	:see-if-stalled, :move-to, :sense, :retrieve, :grasp object, :move, :help-with obstacle :after :set-blocking-obstacles

Table 1: Creature Objects and Operations

SENSORS

Sensors include a proximity sensor, analogous to a ringed set of ultra-sonic sensors; and an imaging sensor.Circumferential sound sensors are available with at least two low-cost commercial platforms (see (4), (5)). The vision system is a line-of-sight stereo system which yields range over a 2D matrix. In the experiments, only beacon creatures have vision sensors and cannot use them when the platforms are in motion.

Proximity Sensor--The proximity sensor has a range-factor which is a multiplier used with a creature's platform radius to get the range capability of the sensor ring. That a larger creature would have a greater proximity range seems intuitively correct (a more massive object needs more of a safety margin, speeds being equal).

The :sense-from operation simulates the action of the ring by extending a ray from the current platform coordinates out to the range limit, around the associated-creature's perimeter at 1 degree increments, and recording the angle to the nearest

obstruction. This simple algorithm was adequate for this study and has been since successfully implemented on a Hero 2000.

Vision Sensor--The vision sensor object has a limiting range. The :locate operation simulates a clockwise scan of the work area from the platform to locate and recognize an object. For each object within range, a LOS determination is made, and if positive, a "recognition" is made. Recognition is assumed to be a simple discrimination among classes of objects known to be in the environment. If the view is obstructed, the obstructing object is recognized, and its name is returned. If the object is recognized and it is the sought-object, then a ranging is made and the object name and its coordinates are returned. Otherwise the process continues until the whole work-area has been scanned, returning nil if unsuccessful.

The vision sensor can be asked to identify an object at or near a certain location. The :identify-object-at operation then simply trains the sensor on the given coordinates and, if unobstructed, returns the identity of the object nearest that location.

CREATURES

The generic creature has a shape, an identification number, a location, a list of on-board sensors, a name, and a sleep toggle for ignoring certain commands. The cc-radius is the radius of the equivalent circumscribing circle. For round creatures this is just the radius; for other objects, an equivalent radius is computed at instantiation. The :sleep and :wakeup operations set and reset the sleep variable.

Round-creature--We begin with creatures whose notions of motion are direction and speed. We believe many of the key ideas embodied in this report extend in a straight-forward manner to a space-based environment. The round-creature is a circular platform capable of omni-directional traverse. The creature has an organic 16-bit processor and 512k bytes of memory, which should be enough computing capability and memory to carry out the operations described below, and an RF broadcast transceiver capable of data communication.

The round-creature has speed (distance units per clock tick) and direction and can be given a target-direction in which to :wander. Caution-speed is the speed used when an obstacle is encountered, and obstacles are noted and placed into obstacle-memory. The obstacle memory is cleared after every 12 obstacle sensings (a kind of forgetting).

Sensing and Avoiding Obstacles--When the :feel-force operation is invoked in the creature's continuous control loop, the :sense operation is invoked, and if there is a positive sensing, the :runaway operation is invoked. Runaway sets the

direction to 180 degrees opposite the obstacle direction (by invoking a :turn in the new-direction), sets the speed to caution-speed, and executes the :move operation. Move advances the creature caution-speed units along its direction. Thus, if an obstacle or other creature approaches close enough, the creature will move away until :feel-force returns nil.

The :sense operation invokes the :sense-from operation of the creature's proximity sensors if any. When an obstacle is sensed, the token UNKNOWN and the location of the creature at the time are stored in obstacle-memory . Obstacle-memory is used to see if the creature is not making progress (see *Getting Stuck* below).

Locating and Identifying Objects--To locate an object, a round-creature asks its sensors each in turn to :locate the object. Only the vision sensors are capable of this operation. If the creature doesn't have such a sensor, it asks another creature to locate the object, which in turn recursively ask other creatures if necessary. This single request thread prevents simultaneous creature responses as in a broadcast method. Most often working creatures (e.g., retrievers) with only proximity sensors can't identify the object to be located. More typically, the request is to :locate-object-at the location which in turn converts to a call to the creature's vision sensor to :identify-object-at that location. In our typical setup, only beacons have vision sensors, and one of them ends up identifying and locating the object.

Round-creature-with-arm--This creature is typically endowed with a swivel retrieval mechanism: an arm that swivels either freely or under control about the platform axis, can be retracted and extended, and needs only to make contact with the objects to be retrieved (one can imagine simple or complex contact mechanisms from velcro pads to magnetized grippers). The :sense command is as described above but augmented to account for the objects in tow.

This creature is used primarily for retrieving a target-object at a target-location. It keeps track of whether its arm is extended or not, whether it has grasped the target-object and its rate of progress through the distance-and-time recordings. If impeded, the creature remembers the blocking-obstacles and whether help-is-on-the-way (stored as a list of the helper and its target-object). If it is helping another creature, it remembers the creature-being-helped.

Retrieving Objects--Retrieving objects is effected as a reaction plan, which is akin to the concept of a Universal Plan (13), once a target-object has been established by the user or another creature. Table 2 shows a retrieval plan. The preconditions of the plan are checked and the appropriate action is taken on each pass of the control loop.

C'4

Precondition	Action	Comments
(and target-object-grasped old-coords (< (distance coords old-coords) retrieve-speed))	Clear all memory of retrieval data, detach and close arm . If helping someone, clear helping data	Done
(and target-object-grasped (equal old-coords target-location))	move-to target-location	Heading home
target-object-grasped	set target-location to old-coords; if helping A, tell A to :set-blocking-obstacles to nil and to set help-is-on-the-way to nil	Turning for home
(and target-location (<= (distance coords target-location) arm-length) (not (object-usurped target-object)))	extend arm and :grasp object, setting target-object-grasped to true	Grab object
Target-object has a value	:locate target-object, and :move-to target-location	Still trying
No target-object	Stay out of trouble	Creature idle

Table 2: A Reaction Plan For Retrieval

To understand the plan, read up from the "Target-object has a value" precondition. When the creature is commanded to retrieve an object (target-object is set to the object name), it first :locates the target and then executes a :move-to the target-location from its old-coords (the position at which it received the :retrieve command). As soon as it is within an "arm's reach" of the object and the object is not being grasped by another creature, it grasps the object. Then it turns and heads for home, making the target-location the old-coords (move-to looks for a target-location). If the creature is helping another creature, it tells the other creature at this point that it has removed the blocking-obstacle and help is no longer on the way. When the creature is within retrieve-speed of the old-coords, the arm detaches from the object, is retracted and the retrieval is complete.

Since the preconditions are checked during each cycle of the control loop, the reaction plan provides robust operations, allowing for a successful retrieval in the wake of many unforeseen actions, e.g., the object moves, or another creature grabs the object first.

Getting Stuck--While retrieving, if the creature is blocked and help is being provided by another creature, the creature just sits and waits for the help. The obstacles are detected as part of the :move-to operation where the first action taken is a local sensing for obstacles. If any exist, a check is made to see if the creature is being stalled (:see-if-stalled) and then a :runaway is executed. Otherwise, a :move is executed at the resulting direction and speed(:move is the same as described above but taking any towed object into account).

To simply determine whether progress is being made, two cases are modeled: one, where, in a certain time period, the distance to the target-object has not changed appreciably; and two, where there has been a large time lapse since we last invoked the :move-to operation. The first case involves the instances where the creature is executing :move-tos and :runaways, but is not really getting any closer to the object. The second case involves executing continuous :runaways because of many obstacles as in the example above.

Each time a :move-to is invoked, the distance from the object and the time are recorded as lists in distance-and-time. For the first case, the difference between the current distance and the distance achieved the last time a :move-to was invoked is compared to a preset value. For the second case, if the elapsed time since the last invocation of :move-to is greater than a preset value, the creature is considered stuck. The values for this simulation (normal speed = 5 and caution speed =1) were 3 distance units and ten clock ticks respectively.

When :see-if-stalled determines the creature is stuck, the last obstacle recorded in obstacle-memory is considered the blocking-obstacle, and the :after operation for setting the blocking-obstacles sends a ":help-with obstacle" call out to other creatures. In the example, retriever #2 answered the :help-with obstacle call from retriever #3.

CONCLUSIONS AND FUTURE WORK

The robustness of the cooperative becomes more evident the more the simulation is run. Objects to be retrieved can be mischievously moved by the user, obstacles can be put in retrievers' paths, and in all but the most pathological of cases (two retrievers latch onto the same object and have a tug-of-war until the user intervenes), the creatures succeed. The current placement of the beacons is such that with only two beacons set at a diagonal across the work area, over half of the retrievals will be successful.

Two creatures can be sent after the same object with the result that the second creature chases and tracks the object until the the other creature releases it. This behavior is interesting to watch in the simulation but underscores the need for human

guidance. Autonomous control of multiple agents is a current research endeavor. The described simulation experiments indicate that robot cooperatives will succeed well against unforeseen events but only for a time by themselves.

An enormous amount of work in algorithms for robot navigation has been done. Some of which are relatively simple to implement (e.g., (10),(8), and (1)), and we will be investigating them for our systems. In this vein, we plan to give each creature a simple map of the objects and the free space. We will then have the creatures broadcast the results of the retrievals so that all creatures can update their space maps, and thus better use navigation algorithms.

We are now moving the experiments onto actual robot platforms. Several low-cost commercial platforms are available (see for example (7) and (4)). In MITRE's Autonomous Systems Laboratory, we have two Hero 2000 robots for which many of the round-creature-with-arm competences have been implemented, but we are acquiring a more capable platform such as the Denning MRV-3. The robots are being programmed using the REX/GAPPS system (11) from Teleos Research. We plan to mount a stereopsis system which produces ranging information at video frame rates (9) on one of the Heroes (as a beacon); the Denning and the other Hero will be retrievers. Experiments with these systems will help us understand how well the co-op ideas stand the test of real systems in real environments.

REFERENCES

(1) Arkin, Ronald C. Motor Schema-Based Mobile Robot Navigation,in *International Journal of Robotics Research,* Vol 8, No. 4 August 1989. pp 92-112

(2) Bloom, Ben; McGrath, Debra; Sanborn, Jim, A Situated Reasoning Architecture for Space-Based Repair and Replace Tasks, *Proceedings of the Goddard Conference On Space Applications of AI,* NASA Conf Pub #3033, 1989

(3) Brooks, Rodney A, A Robust Layered Control System for a Mobile Robot, *IEEE Journal of Robotics and Automation,* 2(1), March 1986

(4) Brooks, Rodney A Autonomous Mobile Robots, in *AI In the 1980s and Beyond,* Grimson and Patil eds, MIT Press, 1987

(5) *The Hero 2000 Technical Manual,* Heath/Zenith Systems, Benton Harbor, MI 1989

(6) Kaelbling, L. P., An Architecture For Intelligent Reactive Systems, in *The 1986 Workshop on Reasoning About Actions and Plans,* M. Georgeff and A. Lansky eds, Morgan Kaufman, 1986

(7) Long-ji Lin; Mitchell, Tom M.; Philips, Andrew; Simmons, Reid, A Case Study In Robot Exploration, CMU-RI-TR-89-1, CMU Robotics Institute, Jan 1989

(8) Meng, Alex C.-C.; Wand, Marty; Hwang, Vincent S., A Methodology of Map-Guided Autonomous Navigation with Range Sensors in Dynamic Environments, *SPIE 3rd Conf on AI Applications,* 1988

(9) Nishihara, H. K. Practical Real-time Imaging Stereo Matcher, in *Optical Engineering,* 23 (5), 536-545 (Sep/Oct) 1984

(10) Rao, Nagewara S. V., Algorithmic Framework For Learned Robot Navigation in Unknown Terrains, *Computer,* June 1989

(11) Rosenschein, Stanley J. and Kaelbling, Leslie P., The Synthesis of Digital Machines With Provable Epistemic Properties, SRI Technical Note #412, SRI International, Menlo Park, CA 1987

(12) Sacerdoti, Earl D., Plan Generation and Execution For Robotics, SRI TR, 1983

(13) Schoppers, Marcel, Universal Plans For Reactive Robots In Unpredictable Domains, in *Proceedings of IJCAI 10,* 1987