To appear in *Mobile Robots XIV*, SPIE Proceedings Vol. 3838, Boston MA, September 1999.

Use of Mission Roles as a Robotic Tasking Device

David Anhalt[1], John Spofford[1]

Center for Intelligent Systems, Science Applications International Corp.

ABSTRACT

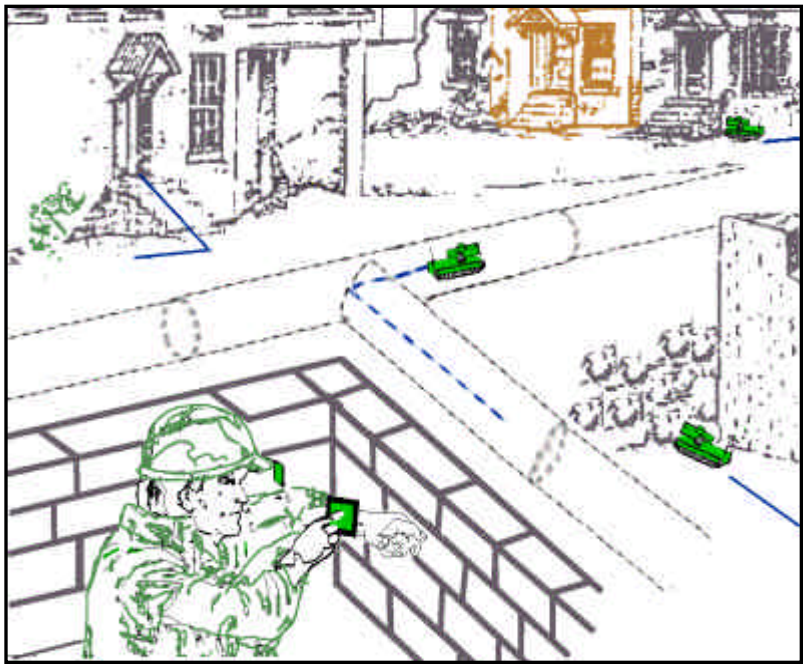
With the increased use of specialized robots within heterogeneous robotic teams, as envisioned by DARPA's Tactical Mobile Robotics program, the task of dynamically assigning work to individual robots becomes more complex and critical to mission success. The team must be able to perform all essential aspects of the mission, deal with dynamic and complex environments, and detect, identify, and compensate for failures and losses within the robotic team. Our mission analysis of targeted military missions has identified single-robot roles, and collaborative (heterobotic) roles for the TMR robots. We define a role as a set of activities or behaviors that accomplish a single, militarily relevant goal. We will present the use of these roles to: 1) identify mobility and other requirements for the individual robotic platforms; 2) rate various robots' efficiency and efficacy for each role; and 3) identify which roles can be performed simultaneously and which cannot. We present a role-base algorithm for tasking heterogeneous robotic teams, and a mechanism for retasking the team when assets are gained or lost.

Keywords: heterogeneous, tactical mobile robots, distributed tasking and control

1. INTRODUCTION

1.1. Tactical Mobile Robotics Overview and Technical Objectives

DARPA's Tactical Mobile Robotics (TMR) program seeks to develop versatile, reactive, and robust teams of small robots to operate in restricted or denied areas under a wide variety of conditions. These robotic teams are being developed specifically for operations in area where humans cannot go due to size constraints, lethal or dangerous environmental factors, or unstable conditions (such as disaster sites).



Using small teams (2 to 10) small robots, working collaboratively and semi-autonomously, the program seeks to have revolutionary impact on the ability of robots to operate successfully in unknown, dynamic, and dangerous environments. In order to achieve this goal, the TMR program is developing several small man-portable robotic platforms, mission payloads, and intuitive, highly mobile user interfaces to control the robotic team. Technologies that must be advanced to achieve the TMR goals include: robotic team configuration and control; robotic collaboration; robust navigation, localization, and mapping; sensor fusion, particular fusing sense data from multiple viewpoints; and target detection and tracking.

[1] danhalt@cis.saic.com, jrs@cis.saic.com

Combining elements from earlier research based on larger robotic platforms[11] and aspects of the behaviorist approach for robot control[3], the TMR program is focused on developing small, backpackable robots to aid military operations and operations other than war [5,2]. A key technology for the TMR program is the development of a control scheme and architecture suitable for tasking and control heterogeneous teams of robots. Of particular interest are the abilities to: 1) the quickly form teams from available robots; 2) control the robots without large amount of *a priori* information about the individual team members; and 3) perform *collaborative* work, where multiple robots work together to achieve a goal .

The control of teams of robots has received considerable research attention. Most research falls into the categories of controlling simulated swarms of robots, Open Agent Architectures[4], reactive and learning systems[6,7] and hybrid systems[1]. Others have controlled team characteristics by simulating tenacity and impatience[9]. Researchers have also been investigating problems associated with close cooperation among robots on tasks such as marsupialism[8]and rendezvousing[10].

1.1.1. Heterogeneous Team Formation & Control

The TMR vision of robust heterogeneous teams of robots performing militarily relevant, autonomous and semi-autonomous tasks requires a tasking and control architecture that can: 1) quickly form available robots into effective robotic teams; 2) control robots with varying abilities, limitations, and payloads; 3) incorporate new *types* of robots, as well as new individuals of known types; 4) deal with battlefield losses and failures, and 5) manage assets to maximize performance based on each robot's available power, endurance, sensor sensitivity, and effector applicability.

In Section 2, we discuss approaches to providing this functionality, emphasizing a "task broker" approach, which allows to system to negotiate with individual robots for the performance of specific tasks.

1.2. Tasking via Roles

Based on mission analysis of proposed missions for the TMR robots, we isolated 47 "roles" that individual robots may perform, and 37 collaborative (heterobotic) roles. We define the word, "Role", as a set of activities or behaviors that accomplish a single, militarily relevant goal. Examples of roles for single robots might include the roles, "Pointman" and "Covert Observer". Heterobotic roles might include "Data Docker" or "Audio Triangulator", roles that require the cooperation of another robot to succeed. Roles are identified by name, and provide key, abstract, *goal-directed* information to the individual robots. Based on the Role, each robot is enabled to achieve the goal in the matter best suited to its own assets and state.

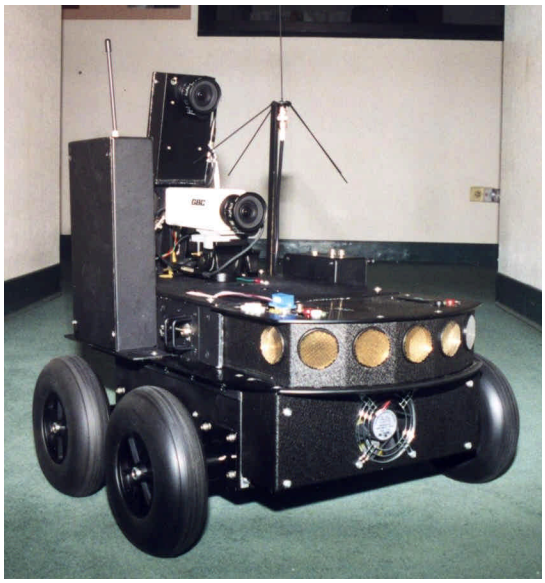


Figure 1. Radbot, Modified Pioneer AT (Packbot Surrogate)

Figure 2. Subot (Throwbot Surrogate)

The concept of Roles is used at multiple levels throughout the architecture. For example, rather than addressing a sensor directly (i.e. through a pointer stored in an instance variable), a behavior asks the Robot to "find" the sensor currently fulfilling the known role, such as "Tracking Camera". By re-finding assets at each instant of usage, the behavior allows the Role of "Tracking Camera" to be moved (for some purpose unknown to the behavior) from one camera to another. Section 3 discusses these concepts and the implications for software and object model design and implementation.

1.3. Example: Role-based Heterobotic Docking

In Section 4, we will step through a docking maneuver, currently under development, performed by a pair of dissimilar TMR surrogates. The TMR program is currently developing two robotic platforms: "Packbot" a high utility backpackable robot, and "Throwbot", a smaller less capable robot that can be thrown, launched or dropped into denied areas. The actual TMR

robots are not yet available, so this research has been performed using surrogate robots. Figure 1 shows a modified Pioneer AT, known as Radbot. Figure 2 shows a small surrogate for the TMR "Throwbot" form-factor, called Subot. Throwbots, due to their ability to be launched or thrown, will be able to reach area denied to larger robots. Once the Throwbot's mission has been completed, it may rendezvous and dock with a larger TMR robot for data and power transfer. This example docking maneuver will first show the system-level control and role assignments, then discuss the robots' responses and execution of the required functionality. The example will also show the decomposition of a Role into several, simpler, context-free behaviors, which collaborate to perform the docking maneuver.

2. HETEROGENEOUS TEAM FORMATION & CONTROL

2.1. Control Architectures for Team Control

The planned tactical use of the TMR robotic team requires that team formation and control be quick, robust, and flexible. In particular, the inclusion of multiple robot types, and the intention of adding new robot types in the future, dictates that the control architecture must be able to incorporate and control robots with minimal impact to the existing system.

There are a number of system approaches to providing these abilities, each with its strengths and weaknesses.

1. Provide detailed knowledge (via download or databases) about the individual robots and robot types to an "executive", which uses the knowledge to control all details of team activity. In this approach, the system executive serves as a surrogate human operator to teleop each robot.
   - − Too much *a priori* knowledge about individuals and individuals' state is required.
   - − Too much communication is required between the system and the individuals.
   - − The vast majority of processing and decision-making must be performed by the executive. This requires significant processing and communications assets for the executive, and creates an easily detectable point of failure.
   - − If communications are lost, individual robots are left helpless and defenseless.
   - + Conforms to traditional, procedural programming model, and matches the human teleoperational model.
   - + High bandwidth status from robots makes dealing with faults and unexpected interactions somewhat easier.
2. Provide only limited control (i.e., "hints") from the higher-level system to the individual robots. Each robot has a number of behavioral directives that control the robot's overall behavior, which are mitigated by the received hints. In this approach, the system serves as a "shepherd" for a group of more-or-less willful robots.
   - − The difficult of estimating *if* and *when* a task will be completed by multiple autonomous actors,
   - − Problems with *efficiency*, due to many actors operating in a (virtually) uncontrolled, uncoordinated fashion.
   - − Difficulty in bringing the right robot with the right abilities to bear on the problem.
   - − The specter of "emergent behavior", in which low-level behaviors combine to cause unexpected responses to stimuli[1].
   - + Programming of robots is limited to the development of relatively simple behaviors.
   - + Redundant response from multiple robots provide a level of fault tolerance.
3. Have the system negotiate, at an abstract level, for services from the member robots. The system tasks the robots with high level tasks, and each individual robot uses its own unique state and assets to achieve the goal. In this approach, the system serves as a broker, assigning tasks based on each team member's abilities to perform the task.

[1] While emergent behavior is a fascinating research topic, proposed users of the TMR system have expressed concern about introducing *more* uncertainty into the battlefield. At this early stage of robot introduction into land warfare, the goal is to provide known, deterministic robotic response to mission directives.

- − This approach requires a means of tasking without reference to individual robotic assets or abilities
- − The limited knowledge at the system (planning) level requires interrogation of the robotic assets to determine their applicability to a task.
- + Allows individual robots to achieve goals with the approach that best suits their assets and state.
- + Enables incorporation of new robot types without detailed knowledge about their abilities and weaknesses
- + Provides layers of information hiding and reduces information transfer during team configuration
- + Provides a mechanism for detecting the best available actor for a task, and for (re)assigning tasks to other team members upon task failure or asset loss.

It is our contention that the task broker metaphor is best suited for a heterogeneous, dynamic battlefield robotic team. The added complexity of system control can be offset, in part, by the use of OOA/OOD and overloading key functions within each robot type's software class. Our TMR system concept uses Roles as the currency within a task brokering system.

2.2. Heterogeneous Robot Team Formation

The rapid formation of teams requires some mechanism that enables the system to build an object model of the team hardware. Information required includes:

- · the type of robotic platforms, and information about their capabilities (i.e., max. speed, range, weight)
- · the type of sensors, effectors, and payloads onboard and their location onboard, sensor models, etc.
- · the role assignments that the sensors, effectors, and payloads will be assign at startup (e.g., TELEOP_CAMERA).
- · the onboard behaviors, and the behaviors' startup roles (e.g., DOCKING_SCANNER)
- · physical dimensions, weight
- · not currently includes, but expected to be added, are wire-frame models of the platforms and components to be used for spatial reasoning and model-based team member recognition

To support reasoning about logistics and team operational limits, the configuration also include power requirements and connectivity (power sources and sinks), information about model and serial numbers, and sources of replacements and parts.

Ultimately, the creation of this team configuration will be performed using data-docking maneuvers. In other words, the robots, upon request, will download to the system all information required to effectively task and control themselves. Currently, this functionality is being simulated using a tree of configuration files. A known base configuration file indicates what types – and how many – robots will be in the team. Each robot, in turn, has a configuration file that provides information about itself, and indicates what types of sensors, effectors, payloads, and behaviors are onboard.

2.3. Issues in Tasking Heterogeneous Robot Teams

When a number of different robot types, with different form-factors, sensor, and payloads, are formed into a team, the system controller cannot specify *how* the tasks will be accomplished. In a role-based system, the top-level controller only specifies what role the individual robots will play. It is up to the individual robots to accomplish those roles.

Unless the higher-level control can understand *all* of the robots' layout, resources, state, and responses, it is impossible for the higher-level control to manage all the details of each robot's response. Simple "hardwired" directives, that single–robot and homogeneous systems are based on become meaningless in heterobotic teams. "Run the corridor-following behavior using the Sony camera on the right fender" cannot succeed if the robot has no fender, no camera, or no corridor-following behavior.

What it means to accomplish a task may also be dependent on the robot performing the task. For instance, an emergency command to "Hide" may mean "Move to a different floor" for a large robot, "move under a table or desk" to a medium sized robot, or "climb the curtain" for a small wall-climber.

2.3.1. Competency-based Response to Tasking

If each robot in the team was tasked to look in a specific direction, the response of the various robots might be quit different. A complex robot with multiple sensors might simply turn a pan-tilt in the specified direction. A smaller robot with a single, fixed sensor would have to move (rotate) in order to turn its sensor to the specified direction.

The more complex robot may be able to continue previously assigned vision-based roles because it could simple turn *one* of its sensor in the new direction. The same command would cause the smaller robot to have a conflict between existing vision-based roles and the new request.

This example shows that even determining which roles conflict with each other may be robot-dependent. Some roles are obvious, clear-cut conflicts. One cannot be an Announcer while being a Hidden Observer, but can a robot be a Occupant Detector while being a Pathfinder? It depends on the sensor loadout and computational capacity of the robot. Therefore, the only safe way for the higher-level system to task the robots is to *ask* them if they can perform the role.

2.3.2. Intra-team Competition for Tasks

Freed from the details of exactly *how* each robot will achieve its goals, the system now must solve a more abstract problem: how to best allocate tasks (roles) to the available robots.

Since the system does not know the details of the robots' configuration, nor the details of the robots' plans for achieve their *existing* roles, it cannot achieve efficiency in role assignments by itself. Instead, the system must ask the robots if the proposed new role conflicts with existing roles on a robot-by-robot basis. As development progresses, we plan to implement a competitive scheme for role assignments that allow each robot to rate itself with respect to the proposed role. When the robots have each submitted a self-rating for the role, the system can then assign the role to the robot that is best suited (under the current circumstances) for the role. Section 3.4 discusses a phased development plan for role competition.

2.3.3. Team Asset Gain and Loss

In battlefield scenarios, it is inevitable that robots will be lost or damaged. The TMR system must be able to detect and recover from loss of robots and robot assets. Critical roles can be assigned and appropriate watchdog timers and status streams can be established. If the goal is not achieved within the allocated time/resource allocation, the system has the option reassigning the role to another robot. The reassignment may be random, based on *a priori* knowledge of the robots' locations and states, or it may be competed to find the best candidate for the assignment.

In addition to handling losses, a robust robotic team should be able to reconfigure in the battlefield to capitalize on "found" resources. For instance, as a conflict progresses, and robotic loses are sustained, several TMR systems may combine their remain robotic resources into a single team. This is equivalent to combining several decimated military units into a single operational unit. We envision a capability where "stranger" robots are placed proximity of each other and (electronically) told "You're a team. He's Pointman. You're Rear Guard." and have the two robots perform successfully as a team from that point.

The data docking form of team configuration is critical for this level of team adaptability. The relatively small amount of information that must be transferred to configure a new robot into a TMR System is a reasonable amount to transfer over a low-power RF or IR data dock link. This will allow unknown robots and unknown robot *types* to be incorporated into a TMR team in the field without software modification.

3. TASKING VIA ROLES

3.1. Mission Analysis & Role Definition

Requirements for the TMR System were generated from analysis of proposed TMR missions and scenarios generated by the TMR Concept Development Group (CDG) and from mission discussions at TMR Quarterly Progress Review. The missions and scenarios that were proposed for the TMR system were analyzed for system and component requirements and for Roles that the robots perform during the missions. The possible roles performed were divided into essential roles and optional/opportunistic roles, that is, roles that may improve performance or probability of success in some circumstance.

Forty-one single-robot roles and thirty-eight collaborative roles have been defined from the mission analysis. We do not consider this an exhaustive list from the current set of missions, nor do we consider the current set of proposed mission to be exhaustive. However, these role provide a sufficiently rich and diverse set to indicate an operational envelope for the TMR System.

The scope and complexity of the defined roles varies greatly. For instance, the role, Announcer, requires only that the robot provide remote public address facilities for the user. At the other end of the complexity spectrum, military roles, such as Scout, may become arbitrarily complex, depending on the terrain, threat, and area of operations.

3.1.1. Single Robot Roles

Single robot roles provide information or behavior required for the successful operation of the robot (as an individual or as a member of a team). These role are typically support autonomy and require limited (or no) interaction with other robots or the user, beyond starting and stopping. Examples of single robot roles include Waypoint Follower and Self Protector. Seven single robot roles have been defined.

3.1.2. Human / Robot Information Sharing Roles

Table 2 shows the current list of roles associated with information collection and sharing between the robot and the user. Human / robot information sharing roles provide information or other services directly to the user, with limited or no interaction with other robots. Forty such roles have been defined. These include roles such as, Smoke Layer, Patroller, Distractor, and Breacher.

3.1.3. Collaborative Robotic Roles

Thirty-seven collaborative roles have been identified. These roles require the active participation and coordinated efforts of two or more robots. While some role do not require physical cooperation (e.g., Auditory Triangulator and Map Sharer), most collaborative roles require that two or more robots meet and work together on some physical task. Examples of these roles include Marsupial Carrier, and Power Docker.

3.2. Role-based Team Configuration

Roles can be assigned at startup by adding them to the robot's (or robot component's) configuration data. In the current configuration file-tree implementation, a set of text lines in the appropriate configuration file, such as:

```
functional_name = POINTMAN
functional_name = THROWBOT_LAUNCHER
```

causes these roles to be launched at startup, and enables other team members to reference the robot by those role names:

```
launcher = Find(THROWBOT_LAUNCHER);
launcher->launch( direction_vector, target_distance);
```

Using Find() hides *all* details of intra-object communication. This allows the communications and underlying data objects to be modified without impacting operational code. For instance, for off-board assets, Find() may use an underlying CORBA naming service, or other communications protocols. Likewise, the underlying local data structures (currently most assets are stored in short linked lists) can be change to hash tables or other data structures as required.

3.3. Software Implications

3.3.1. Overview of the TMR Object Model

Figure 3 shows a simplified representation of the TMR Object Model. Note that all classes are rooted on one of two abstract classes, MRPrimitiveObject or MRObject. The primary difference between the two trees is that the MRObject-rooted classes inherit the team configuration and Find() protocols. All subclasses of MRObject participate in the startup process (see next section), and inherit the configuration functionality – and most of the configuration file data parsing, from their superclasses.

3.3.2. Context-Free Object Protocols

Since roles may move without any notification to the users of resources, objects cannot assume that historical data and object references are still valid. For this reason, object that require information or services from other object must Find() them *at each instance of usage.* This significantly effects the design of objects. Instance variables within class definitions are used only for *local* date (i.e., *not* for pointers to other objects). Pointers to collaborating objects are fetch using Find() at the moment that they are needed.

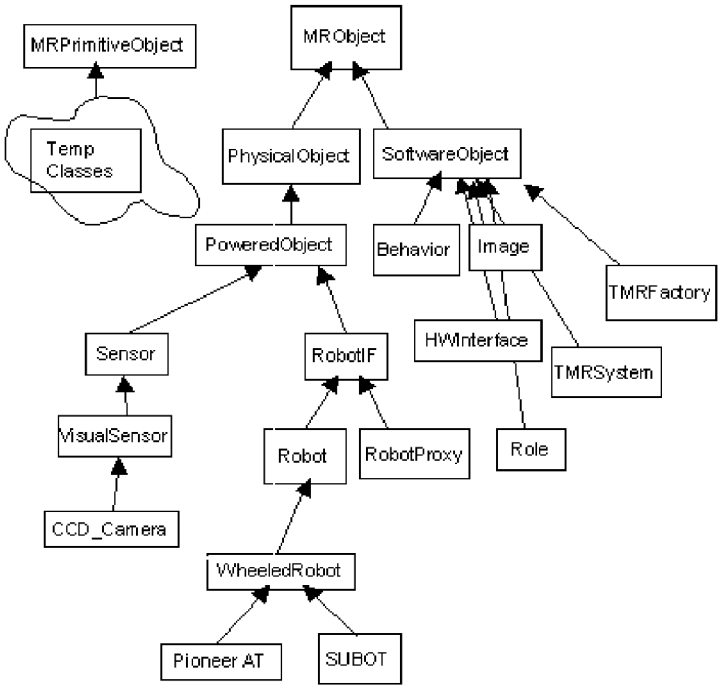


Figure 3. Simplified TMR Object Model.

3.3.3. A Role-based Function Template

The necessity of fetching new pointers to resources and collaborating objects at the instant of usage causes virtually all significant functions to conform to a single structure. A pseudo-code version of this structure would look like:

```
any_func( string resource_role_name, string collaborator_role_name, …)
{
        Find() required resource(s), if any
        Find() the collaborator(s), if any
        use local data, the resources, and collaborators to
                perform the work
        Find() the downstream users of the results
        send the results to the downstream users
}
```

While this may seem tedious and computationally expensive, Find() typically only requires the traversal a single short linked list or a single access to a naming service. The benefit for the effort is that behaviors and other objects are context-free, making no assumptions about what resources or other assets are available. Thus, a set of docking behaviors that work on a tracked vehicle with multiple sensors can also run on a two-wheeled robot with a single camera, without any software changes. See Section 4 for an example of how this basic template is used within the physical-docking task.

3.4. Phasing in Competitive Role Assignment

The ability to "compete" the assignment of roles among available resources will add significantly to the robustness and adaptability of the TMR system. A role is competed by asking the robotic team which individual would be best suited for a

task. After each robot has responded by returning its "score" for the task, the system can then assign the role to the winning individual. For example, competition among resources can be used to improve individual and team performance in the following scenarios:

- When the system is tasked to enter a building, all applicable robots might be assigned the role, "ENTRYPOINT_DETECTOR", which would cause them all to look for an entrypoint into the building. The robot that finds the first (or best) entrypoint "wins".
- When a surveilled object is about to be lost by a tracking robot (due to area-of-operations restrictions, inability to match the object's speed, or obstacle problems), the system can compete the other robots to find a robot that can detect the tracked object and can assume the tracking task.
- If an image of a particular object or location -- possible from a specific viewpoint -- is needed, the system can compete the task to find the robot that has the right sensor and is in a location that reduces the work of capturing the image.

The quality of the competition depends on the level of effort put into the robot's analysis of the task and the depth of reasoning applied to the scoring. At the simple end of the spectrum, a simple Boolean response indicating whether the robot supports the roles can indicate whether a robot can support a role. This request can be handled by the robot's local proxy without any communication with the actual robot. On the other end, a detailed analysis, taking into account the robot's location, fuel state, payload, and concurrent roles, requires considerable communications with the robot, but should result in a better assessment of the situation. Table 2 indicates the levels of sophistication and performers that are required for various levels of competition. It is our intent to gradually phase-in the higher level of competition as the program continues.

Table 2. Role Competition Phase Plan.

Phase	Performer	Results
0 -- hardcoded assignments	System / User	N/A
1 – role support	Robot Proxy	Boolean (can / cannot perform role)
2 – Phase 1 + role conflict detection	Robot Proxy	Boolean (can / cannot perform role), name of conflicting role, if any
3 – Phase 2 + resource applicability	Robot	Variable score, based on robot & robot resource applicability
4 – Phase 3 + location and state scoring	Robot	Variable score, taking into account the robot's resources, location, fuel state, etc.

4. EXAMPLE: ROLE-BASED HETEROBOTIC DOCKING

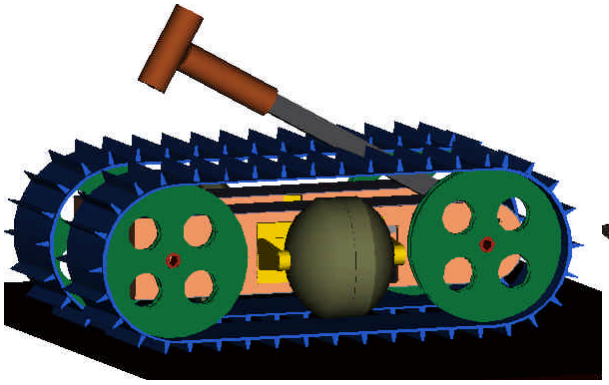


Figure 4. Track Well Marsupial Carrier

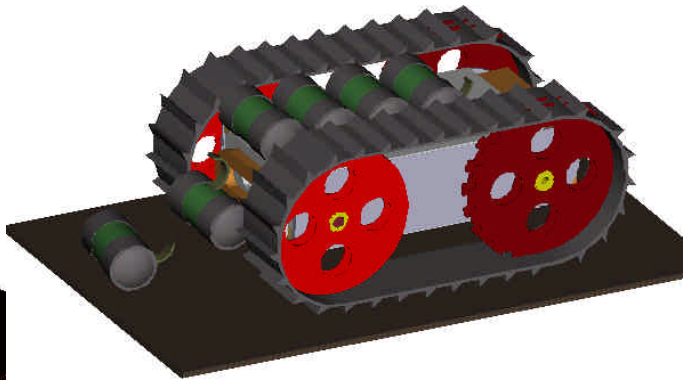


Figure 5. Multiple Marsupial Carrying

4.1. Docker" and "Dockee" Role Definition

The Docker role provides the mechanical and software tools for physically docking another robot. This is a basic maneuver that serves as a key functionality to be used in mission scenarios for high-security data transfer, power transfer, the transfer of physical samples, or marsupialism. Figure 4 shows one concept for marsupial carrying of a Subot by a Packbot-sized robot. Figure 5 shows a concept for multiple marsupial carrying. Physical docking is image-based, where 2D sensing (video) is used to perceive the location and pose of the two participants with respect to each other.The robot assigned the Docker role is the

controlling entity during the maneuver. The Docker may move itself, direct the Dockee to move, or both. Due to the image processing component of the process (estimating pose and location of the Dockee), the more computationally capable robot would most likely be assigned the Docker role.

The Dockee role provides the "other side" of the docking operation. Typically, the Dockee is a slave to the Dockee, moving only as directed by the Docker. In some cases, the Dockee's sensors may be better (or better positioned) for some phase of the approach. In this case, the Docker may task the Dockee to provide it with video frames from its sensor, or the Dockee/Docker roles may be reversed, allowing the (former) Dockee to control the relative motion during that phase of the maneuver.

To successfully control the docking maneuver, the Docker must

1. Detect the Dockee
2. Estimate the Dockee's location and pose
3. Either
    a) approach the Dockee, or
    b) control the Dockee to approach the Docker, or
    c) move both participants toward a meeting point.
4. Perform the final (physical) docking.

In the rest of Section 4, we will discuss how this is accomplished, based on role assignment, and how those assignments are made.

4.2. "Docker" On-Board Behavioral Configuration

The Docker's task was decomposed as requiring five behaviors or behavior types.

Scanning Behavior[2]. This behavior scans its designate field-of-regard (FOR) trying to detect and identify the Dockee. If the FOR is larger than the sensor's horizontal field of view, the robot must make angular moves of the sensor until the FOR has been covered. This may be accomplished by panning a pan-tilt or by physically rotating the robot itself.

Detect/ID Behavior. This behavior is used to detect and identify the Dockee. It is used by the Scanning, Approach, and Docking behaviors to find and track the Dockee.

Docking Approach Behavior. This behavior moves one or both participants into close proximity for the final docking maneuver.

Physical Docking Behavior. This behavior is responsible for the "end-game" final physical docking.

These behaviors are instantiated on robots capable of serving as Dockers during the team configuration by including lines similar to the following in the robot's configuration file:

```
component_type = behavior
config_file = .\\RADBOT\\behavior\\docking_scanner.cfg
```

Each behavior included into the robot configuration is provided with a configuration file, which may be as simple as:

```
TYPE = docking_scanner
name = docking_scanner_behavior
field_of_regard      = 180.0
component_role = DOCKING_SCANNER
```

Which sets its default field of regard (180 degrees) and assigns it the role of Docking Scanner.

4.3. The Docking Maneuver

A docking maneuver is instigated from a plan or from actions of the user. In this example, we will follow the docking control of a Packbot surrogate, Radbot, with a Throwbots surrogate, Subot. The process starts when a specific waypoint is reached or a button is press by the user. The system performs the following operations:

---

[2] This scenario assumes that the two participating robots are in each other's field of regard (i.e., within the field of view of sensors if the sensors are scanned).

radbot = Find(“RADBOT”);

subot = Find(“SUBOT”)

subot->AddRole(“DOCKEE”, “RADBOT”);

radbot->AddRole(“DOCKER”, “SUBOT”, “DOCKER_DETECT_ID”, “SAPHIRA_MOBILITY_BEHAVIOR”);

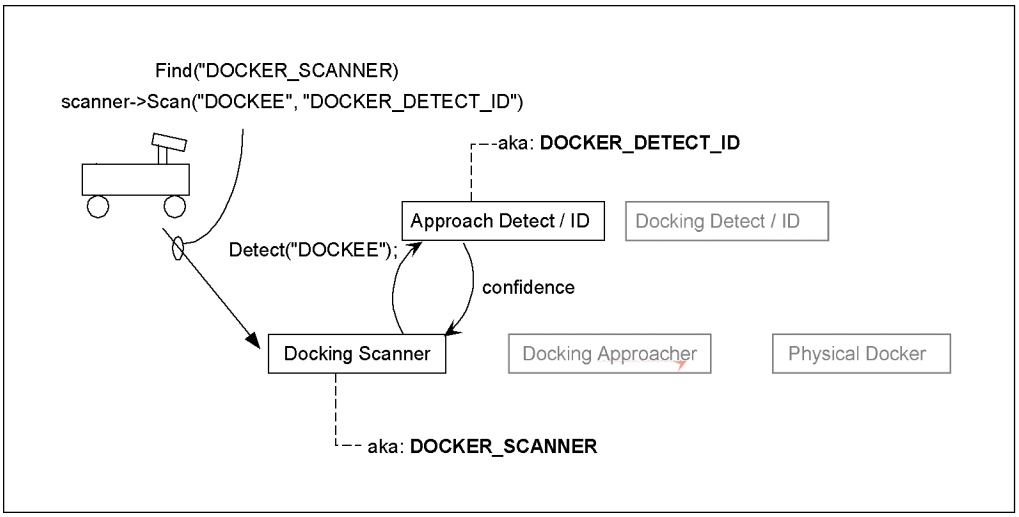


Figure 6. Startup of Docking Maneuver.

In response to having the Docker role added, Radbot tells one of its vision-based object detection behaviors to assume the role of DOCKING_DETECT_ID:

detector = Find(“APPROACH_ID_BEHAVIOR”);

detector->AddRoll(“DOCKING_DETECT_ID”,);

Once the detection behavior has been assigned its role, Radbot initiates the dock by telling the scanner to scan for Subot:

scanner = Find(“DOCK_SCANNER”);

scanner->Scan(“DOCKEE”, “DOCKING_DETECT_ID”, “MOBILITY_BEHAVIOR”);

When the scanning behavior receives this command, it first finds the detection behavior, and the interface that it will use to move the vehicle:

detecter = Find(“DOCKING_DETECT_ID”);

mover = Find(“MOBILITY_BEHAVIOR”);

Note that the reference uses the *role name*, not the behavior’s name. This allows any number of behaviors to serve the role of DOCKING_DETECT_ID. For instance, different detect behaviors might be used in different lighting and weather conditions, when differnet types of objects are being detected, etc.

The Scanning Behavior knows nothing about the sensor or how the detection behavior works (beyond it’s public interface). However, in order to calculate its scanning moves, it must know the horizontal field of view of the detection sensor, so it asks the detection behavior to Find() it, the asks it for its HFOV:

camera = detector->Find(“DOCKING_CAMERA”);

hfov = camera->GetHFOV();

Once the number and magnitude of scanning moves has been calculated, the scanner moves to the first position, and asks the detector to try to detect the Dockee:

confidence = detector->Detect(“DOCKEE”);

The name of the target object is passed in so that the detector can reference system-level information about the target (e.g., models, recognition schemes), or may communicate with the Dockee to perform recognition handshaking (passing of

passwords, flashing LEDs, etc.). The detector returns a confidence metric back to the scanner, indicating whether it detected the Dockee or not. Figure 6 shows the object interactions during this phase.

When the scanner has completed its scan (or has found the Dockee with such high confidence that the scan can be terminated), the scanner re-orients the sensor to the detected object, then calls upon the approach behavior to control the approach:

approacher = Find("DOCK_APPROACHER");

approacher = Approach("DOCKEE", "DOCK_DETECT_ID", "MOBILITY_BEHAVIOR");

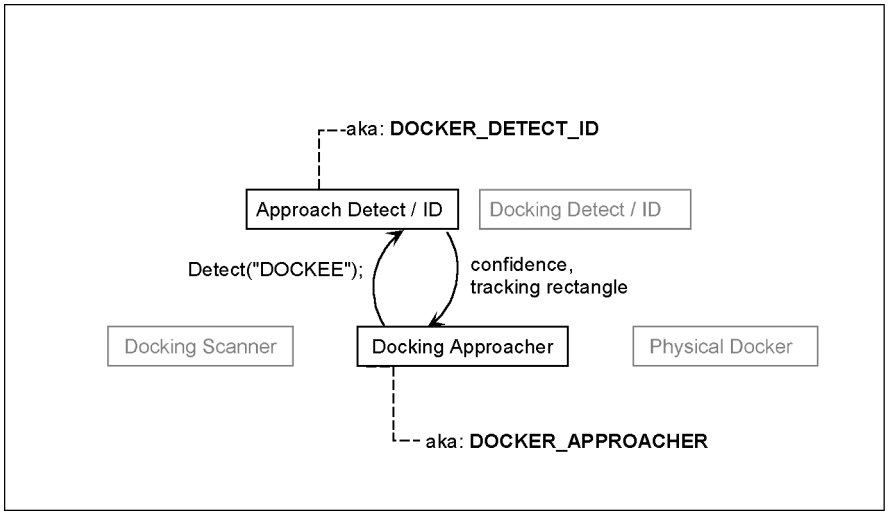


Figure 7. Early Approach Phase of Docking Maneuver.

Figure 7 shows the interaction between the approach behavior and the detection behavior.

Figure 8 shows the configuration after the transition of the DOCKER_DETECT_ID role to the new detection behavior. Using the same code as before, the approach behavior can now finish the approach, positions the robot for final docking. When the position is reached, the final docking behavior is used:

docker = Find("DOCKER_DOCKER");

docker->Dock("DOCKEE", "DOCKER_DETECT_ID", "MOBILITY_BEHAVIOR");

Figure 9 shows the interactions during the final, physical docking phase. The physical docking behavior then performs the final docking.

5. CONCLUSION

This paper has presented a method of tasking and controlling heterobotic teams of robots using mission-oriented *Roles* as the primary tasking mechanism. The use of roles provides a direct link between the mission goals and the robotic software used to achieve those goals. Roles reduce the amount of *a priori* information that the system and member robots must know about each other. The use of roles allows each robot to fulfill its roles as best suited to its own design, assets, and state. The use of Roles enables the system to dynamically re-assign resources (robots, sensors, and effectors) and tasks without effecting other ongoing behaviors. By dynamically reassigning roles to assets, the system can modify the behavior of the member robots, and the behavior of the team as a whole. The use of Roles also simplifies team configuration, the handling of battlefield loses and failures, and the incorporation of new robots and robot types into the team.

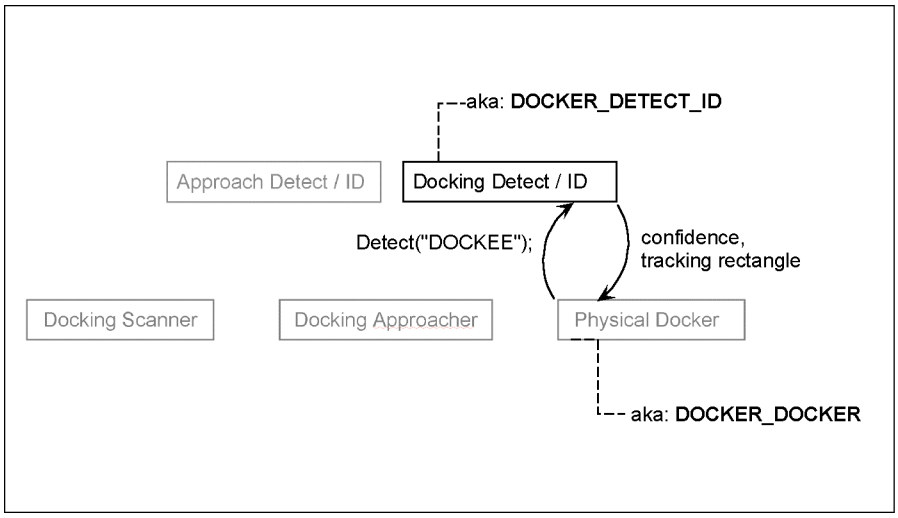


Figure 9. Final, Physical Docking Phase.

ACKNOWLEDGEMENTS

This work was sponsored by DARPA under contract number DAAE04-98-C-L037, administered by U.S. Army Tank-automotive and Armaments Command. The robot testbeds were developed under SAIC IR&D projects 0839 and 0869.

REFERENCES

1. R. Arkin, *Behavior-Based Robotics*, MIT Press, Cambridge, MA, 1998.
2. J. Blitch, "Artificial Intelligence Technologies for Robot Assisted Urban Search and Rescue," in *Expert Systems with Applications*, May 1996.
3. R. Brooks, "A Robust Layered Control System for a Mobile Robot", IEEE Trans. Robotics & Automation 2, #1, pp.14-23, 1986
4. D. Gussoni, A. Cheyer, L. Julia, K. Konolidge, "Many Robots Make Short Work", *AI Magazine*, **18**(1), pp. 55-64 (Spring 1997).
5. E. Krotkov, and J. Blitch, "The Tactical Mobile Robotics Program," to appear in the *International Journal of Robotics and Automation*, June 1999.
6. M. Mataric, "Interaction and Intelligent Behavior," Ph.D. thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, May 1994.
7. M. Materic, "Designing and Understanding Adaptive Group Behavior", *Adaptive Behavior*, 4:1, December 1995, 51-80
8. R. Murphy, M. Ausmus, M. Bugajska, T. Ellis, T. Johnson, N. Kelley, J. Kiefer, L. Pollock, "Marsupial-like Mobile Robot Societies," *Agents 99*, pp. 364-365, Seattle, WA, May 1999.
9. L. Parker, "On Design of Behavior-based Multi-robot Teams", *Advanced Robotics*, 10:6, pp 547-578
10. N. Roy, and G. Dudek, "Learning to Rendezvous during Multi-agent Exploration," *Proc. of the Sixth European Workshop on Learning Robots (EWLR-6)*, Brighton, UK. August 1997.
11. J. Spofford, R. Rimey, and S. Munkeby, "Overview of the UGV / Demo II Program," in *Reconnaissance, Surveillance, and Target Acquisition for the Unmanned Ground Vehicle: Providing Surveillance 'Eyes' for an Autonomous Vehicle*, Editors O. Firschein and T.M. Strat, Morgan Kaufmann, pp. 21-40, 1997.