

Cooperative Control of a Semi-Autonomous Mobile Robot

Jonathan Connell and Paul Viola

IBM T.J. Watson Research Center
P. O. Box 704
Yorktown Heights NY 10598

ABSTRACT

Consider the differences between riding a horse and driving an automobile. A horse will not run into a telephone pole at high speed. If you fall asleep in the saddle, a horse will continue to follow the path it is on. If you have two horses but only one rider, you simply have one horse follow the other. In general, horses are much smarter than automobiles and thus provide a better model for control of a mobile robot. In this paper we describe an actual robot built along these lines, and show how supervisory control can be added to a reactive multi-agent system.

INTRODUCTION

Recently, multi-agents control systems for robots have become popular. Representative of such system is Brooks's subsumption architecture [2] in which large collections of individual behavioral modules fight it out for control of the robot's effectors. Typically, however, there has been no way to externally guide such robots; they are, in essence, "artificial creatures" which autonomously perform only the particular tasks for which they were designed. Yet this does not mean they are totally immune to outside influences. For instance, a collision-avoiding robot can be "herded" in various directions by walking closely behind it. Still, in certain cases it would be convenient to have more direct control of the robot.

There are a number of methods that can be used to achieve this within the multi-agent framework. The most obvious is to cast the human or higher level program as a agent itself which injects commands directly into the arbitration network (e.g. [3]). A different approach is to treat the existing behavioral agents as simply an enhanced effector command language, and design a higher-level control structure that switches them on and off. Indirect methods similar to this have been suggested by several other researchers [5,1]. The robot we describe here takes advantage of both this switching idea and the injection idea.

The actual robot, "Mister Ed", was built to mimic some of the abilities of a horse, as discussed above. Figure 1 shows a picture of the vehicle complete with passenger seat. The user sits in this chair and can directly guide the robot via a hand-held joystick. Also, below the seat there is a bank of toggle switches that allow the rider to authorize the robot to perform certain tasks autonomously. These activities include steering around obstacles, traversing hallways, turning at doors, and following other moving objects.



Figure 1. Our mobile robot has a seat for a passenger. To steer the vehicle the driver can directly interact with the navigation sensors, make suggestions through the joystick, or authorize various activities using a bank of toggle switches.

THE BASIC ROBOT

The control system for our robot consists of a collection of situation-action rules. While production systems have been found inadequate for general-purpose reasoning, they work admirably in the robotics domain. This is perhaps because the rules we use are in propositional versus predicate logic. That is, we never have a situation in which we have to bind X to OBSTACLE-237. With large rule bases unification of variable-laden antecedent and consequent clauses can become time-consuming, and intelligently searching through chains of such rules can be difficult. Our approach is to instead use an "indexical" style of programming which makes reference to individuals like the-thing-currently-blocking-the-path. Such items are defined naturally by the situation the robot finds itself in, and can be used, much like variables, to allow the robot to react similarly to a variety of different instances [6]. The expressive power of this method is

sufficient to allow fairly complex behaviors to be implemented (see [4]). However, like most production systems, the lack of internal structure leaves the system open to unforeseen dependencies between rules.

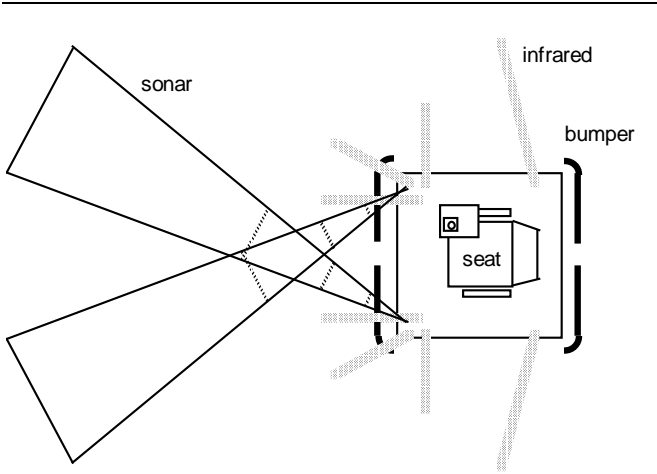


Figure 2. As shown in this overhead view, our mobile robot has 3 sensory modalities. The infrared proximity sensors are used for local navigation and obstacle avoidance, the sonars are used for following people, and the bumpers are used for collision detection.

The input to our production system controller comes from a series of sensors mounted around the periphery of the robot. As shown in figure 2, the robot obtains information from three different sensory modalities. At the front and rear of the robot there are bumper switches (black bands) to detect any collisions that may occur. Each bumper is divided in half to allow lateralization of the contact. For local navigation there are 8 Banner proximity detectors arrayed around the robot (gray lines). These units ("IRs") emit infrared energy in a tight beam and look for any reflection above a certain intensity threshold. The sensors have been angled downwards to insure that any object sensed is actually between the robot and the floor intersection point. Finally, there are two Polaroid sonar sensors mounted at the front of the robot such that their radiation patterns overlap. The echoes from each side are classified into one of 4 distinct qualitative regions (dotted lines show divisions) to assist the robot in following objects.

The production rules control the motors on a two-wheeled TRC Labmate base. This platform is approximately 30 inches square and is capable of carrying over 100 pounds of payload. We divide the motion control problem into a forward/backward velocity component and a left/right velocity component. Then, for convenience, we quantize each degree of freedom to only 2 bits - positive, negative, or stopped. A Motorola 68HC11 microprocessor evaluates all the behavioral rules 15 times a second and updates these four motion control bits accordingly.

RULE SYSTEM

It is relatively easy to create a competent rule system for a mobile robot. A summary of the various behavior modules used by our system is given below. These rules are listed in order of ascending priority. For instance, the output (if any) of the **Retreat** rule completely overrides any output produced by the **Trek** rule. Notice that the behaviors which control the robot's translation are separated from those that control its rotation. This is because

sometimes the relative priorities of related behaviors are different. For instance, if the robot hits an obstacle, **Bounce** insists on backing up no matter what the driver commands via **Joy-Trans**. Yet, since **Joy-Turn** takes precedent over **Turn**, the robot is willing to let the operator choose a particular direction of travel by deflecting the joystick. In this way, commands from different behavioral agents are combined in a distributed arbitration network.

Translation behaviors (in order):

- Approach** - Go forward if an object is detected beyond the optimal sonar range.
- Trek** - Go forward if any of the side IRs have been active recently.
- Retreat** - Go backward if an object is detected in the near sonar range.
- Joy-Trans** - Obey all of the driver's forward and backward commands.
- Bounce** - Go forward or backward a while depending on which bumper was hit.
- Halt** - Prevent travel in the direction corresponding to the impacted bumper.
- Fend** - Prevent forward travel if any of the front IRs are active.

Rotation behaviors (in order):

- Align** - Turn toward the the nearest object detected by the sonar.
- Parallel** - Suggest turning to make both side IR sense the wall if one did.
- Turn** - Turn away from side of the recently hit bumper.
- Stymie** - Turn left if all the front IRs are active.
- Thread** - Turn to unblock a single front IRs.
- Veer** - Turn to unblock a single diagonal corner IRs.
- Twist** - Turn toward the side on which a persistent IR signal recently disappeared.
- Joy-Turn** - Obey the driver's left and right turn commands.
- Point** - Prevent any turn if a bumper is hit.

The basic override operator, called *suppression*, can also be decomposed into two more basic interactions which are useful in their own right. An agent can either *suggest* going one way or another, or *prohibit* motion in a particular direction. By simultaneously suggesting a particular action and prohibiting the opposite action, an agent can completely specify a desired motion for the robot. Notice that while most of the rules specify a definite direction, some use the vaguer suggestion and prohibition methods. This incomplete dominance can be useful in a number of situations. Suppose the robot is being driven through a clear area when it suddenly encounters an obstacle dead ahead. In this case, the **Fend** rule prohibits any further forward motion but does not tell the robot to back up. The operator can accomplish this, if he desires, by pulling back on the joystick. However, since the operator's commands are relayed through the lower priority **Joy-Trans** rule, pushing the joystick forward will have no effect.

Similarly, consider the interactions diagrammed below. This arbitration sequence comes from a situation in which the robot is following an object that has just moved to the left. However, there is also happens to be a wall on the robot's right. Thus, **Align** tells the robot to turn left yet **Parallel** suggests going right. As shown, each module specifies MUST (1), MUST-NOT (-1), or DONT-CARE (X) for the relevant control variables. These are then

merged with the previous combined command to yield a turn direction for the vehicle (see right hand column). In this case, the robot splits the difference between the two behaviors and continues going straight. However, if a higher priority rule such as **Veer** is triggered, it will override this compromise command and take complete control of the robot's heading. Of course the highest priority behaviors have the last word. So if the robot happens to collide with something as it is moving around, **Point** can still freeze the robot in place.

Module	L	R	combination method	⇒	L	R	action
Align	1	-1	as the default	⇒	1	0	= left
Parallel	X	1	as a suggestion	⇒	1	1	= straight
Veer	-1	1	as an override	⇒	0	1	= right
Point	-1	-1	as a prohibition	⇒	0	0	= straight

META-LEVEL CONTROL

It usually not desirable to run all the robot's rules at the same time. We have already seen one case in which the conflicting goals of person tracking and wall following interfere with each other (**Parallel** versus **Align**). For this reason, we have segregated the rules into small groups to form a partitioned production system. These groups can be selectively enabled and disabled by a series of operator controlled switches. This is how the operator tells the robot what its current goal should be. While the robot is good at tactical navigation, it needs this strategic level input from the human.

However, each user-level goal is usually comprised of a number of sub-goals. Figure 3 shows this breakdown. Thus, there is generally not a one-to-one mapping between switches and behavior groups. Also, certain basic behaviors, such as the joystick interpretation rules, are always active (dark bordered box) and others are active anytime the robot is allowed any control at all (gray bordered box). In our robot these groups correspond roughly to the universal goals of obedience and self-preservation.

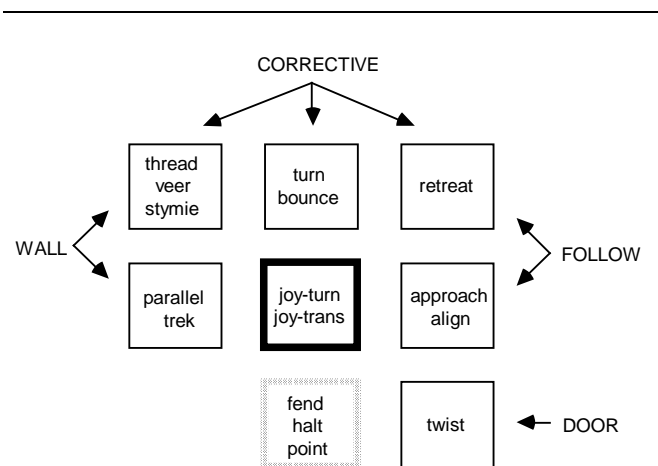


Figure 3. The robot's situation-action rules are broken up into a number of functional groups. The robot can be authorized to perform a specific activity by enabling certain of these groups. Some groups (heavy bordered boxes) are always active, no matter what activity is selected.

The use of this partitioned system is best shown through several examples. Suppose, at the start, that all groups except the joystick repeaters and basic protective behaviors are disabled. The operator is basically free to drive the robot in any direction desired. However, if the front IRs detect an obstacle, the robot will refuse to continue going forward (see figure 4). Like a horse, it balks at this command. In certain cases, such as traversing a narrow doorway, the operator may decide that the robot is being too cautious and then disable the protective behaviors to obtain complete control. On the other hand, he may instead wish to delegate more authority to the robot by enabling the corrective steering behavioral groups. The rider now retains control of most of the forward/backward motion of the robot, but does not have to worry about veering around obstructions. He simply tells the robot to go forward and it takes care of the moment-to-moment heading changes required.

The rider may eventually get tired of driving altogether and give the robot even more control by activating the person following routines. This is useful in large open spaces where the final destination is unknown. Alternatively, the rider may instead decide to travel along some particular environmental defined path, such as a the edge of a room or corridor. To start the robot in some particular direction, the operator enables the appropriate rule groups and then drives the robot until it is properly oriented with respect to the wall or corridor it is supposed to follow. At this point the driver can relax and the robot will continue along this environmentally defined path indefinitely. He is free to fall asleep in the saddle if he wishes. In this case the robot is acting as a smart peripheral. A similar approach could be used to emulate the convenience of the "grab-it" button found in many video-games. By relying on a collection of local procedures, the high level controller need not worry about actually moving the agent's hand or about a achieving a firm grasp on the desired object [4].

When running in the autonomous wall following mode the driver will occasionally resume control for a moment. The usual reason is to a make strategic navigation choice such as which corridor to follow at a branching, or whether to go through an open door. Yet, even some of this can be delegated to the robot. If the door finder routine is enabled the robot will execute a 90 degree turn at the next aperture it sees, and then continue forward using its local navigation routines. Thus, the driver never has to take real-time control of the robot, he merely modifies its reflexes so that the appropriate action will automatically be taken in each situation which is likely to occur. This is analogous to advice such as "the play is to second" muttered during a baseball game. All the computation intensive deliberation can be performed off-line and then compiled down for later use should the contingency arise.

DISCUSSION

Unlike a conventional control program, Mister Ed's architecture lacks strict hierarchical structure. It has been pointed out, in regard to related research, that the homogeneous soup in which rules interact is quite difficult to analyze and understand. This becomes an especially serious concern as the size of the rule base grows. Brooks, in his original work, introduced a mechanism to divide behaviors into levels of complexity. The defining metaphor in this case was evolution: previous levels provided the foundation on which more complicated levels were built. Unfortunately, in practice higher levels often relied on the internal structure of lower levels, thus sacrificing modularity.

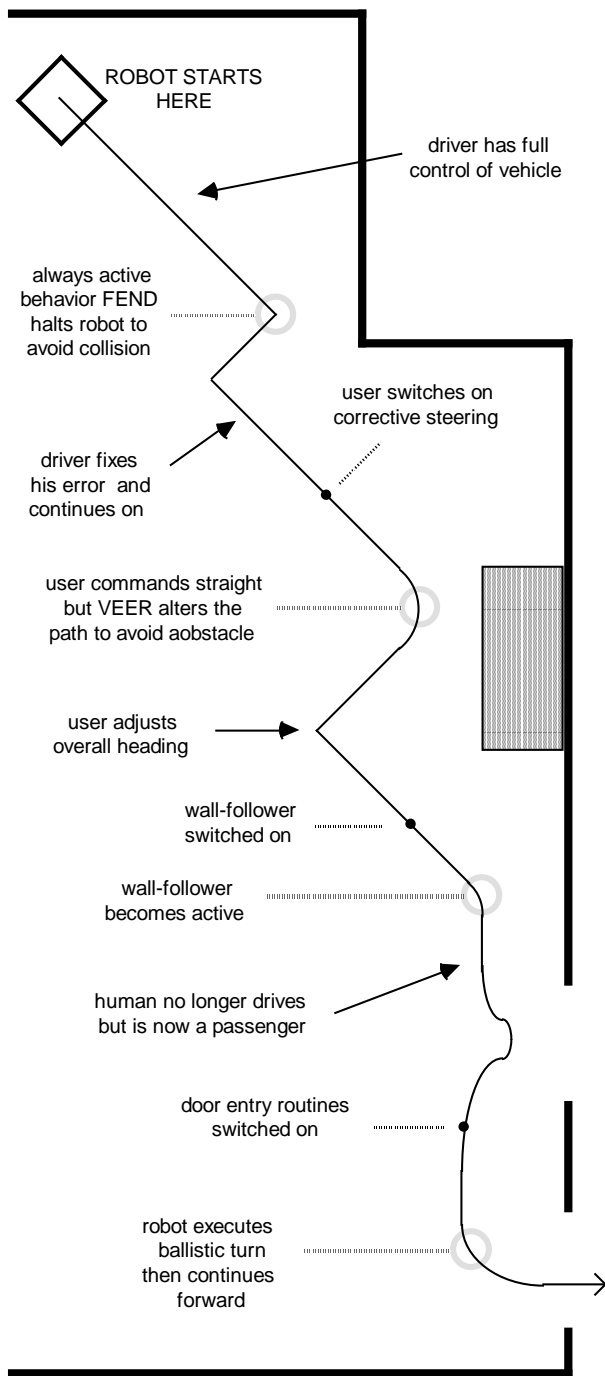


Figure 4. This representative path shows the interaction between the driver and the hardwired control system. As the robot progresses, the user gives it more and more autonomy by switching in new groups of behaviors.

We address this problem by using “competency modules” to reduce the scope of inter-rule dependencies. As described earlier, each small set of rules constitutes a toolbox for achieving a particular goal. Since they are not intended to be used together in arbitrarily large groups, as a first approximation rule interaction analysis need only be performed **within** each competence module. At the next level of abstraction, each such grouping would be described in terms of its ensemble behavior. Further analysis, if required, can then be performed by comparing just these aggregate properties. As the system expands, it may even become desirable to collect several competence modules into a larger chunk such as “indoor navigation procedures”. By imposing such modularity boundaries, we allow the interaction problem to be factored into manageable sized components.

In addition to their theoretical benefits, competence groups also form a natural basis for supervisory control. This is because they are so tightly linked with the creature's goal structure. Our architecture promotes a partnership in which the machine takes care of all the routine work while the person decides what sort of things need to be done. One can imagine a number of applications in which this might be desirable. For instance, a robot such as ours might be used by a hospital patient who did not have the fine motor coordination necessary to continuously drive a normal joystick-controlled wheelchair. For other applications, such as high speed, low flying aircraft, even an un-impaired human's responses are not fast enough and a similar system might be employed for time-critical maneuvering.

On a more down-to-earth level, the particular control system described here would provide a good framework for building a smart camera platform. There is nothing that says that the entity providing overall guidance has to be human. When the cameras see an object of interest they could just tell the robot to head straight for it, trusting the underlying reactive control system to keep the robot on a collision-free path. By decoupling the concurrent goals in this manner, we would be free to focus our attention on the more interesting visual aspects of the problem.

REFERENCES

- [1] Tracy L. Anderson and Max Donath, “A Computational Structure for Enforcing Reactive Behavior in a Mobile Robot”, *Proceedings of the 1988 SPIE Conference on Mobile Robots*, 198-211, 1988.
- [2] Rodney Brooks, “A Layered Intelligent Control System for a Mobile Robot”, *IEEE Journal Robotics and Automation*, RA-2, 14-23, 1986.
- [3] Jonathan Connell, “Navigation by Path Remembering”, *Proceedings of the 1988 SPIE Conference on Mobile Robots*, 383-390, 1988.
- [4] Jonathan Connell, Minimalist Mobile Robotics: A Colony-Style Architecture for an Artificial Creature, Academic Press, Cambridge MA, 1990.
- [5] David Payton, “An Architecture for Reflexive Autonomous Vehicle Control”, *Proceedings of the 1986 IEEE International Conference on Robotics and Automation*, 1838-1845, 1986.
- [6] Devika Subramanian and John Woodfill, “Making Situation Calculus Indexical”, *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, 467-474, 1989.