Autonomous Navigation on a Shoestring Budget

How to transform your mobile platform into an autonomous machine that can self-navigate in an indoor environment.

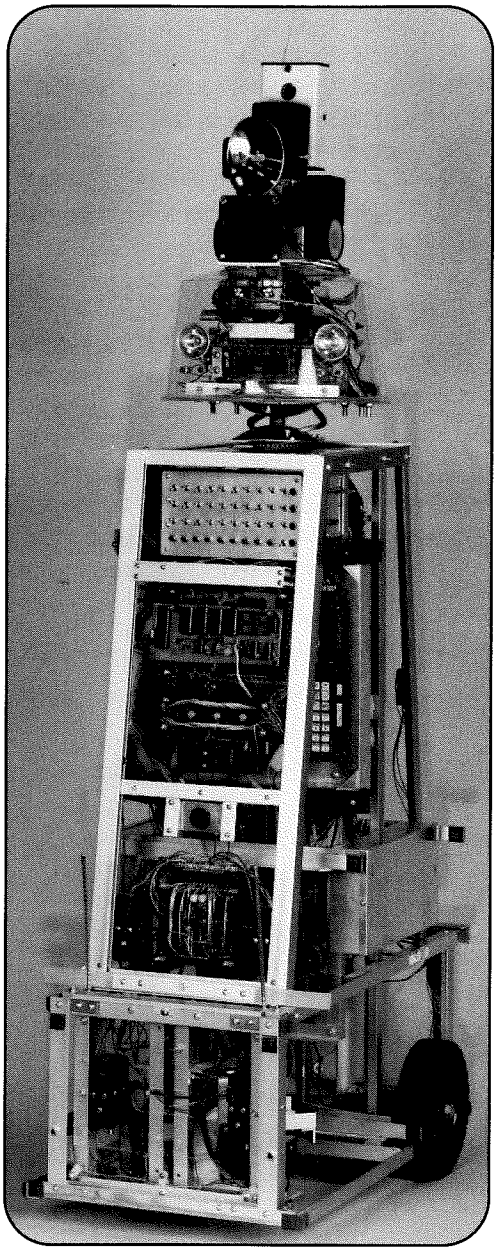

Figure 1: ROBART I *(1980 - 1982) was a fully autonomous interior security robot controlled by a single onboard 6502-based microcomputer.*

by H.R. Everett

There is a definitive milestone in the evolutionary development of an autonomous mobile robot which every would-be creator must eventually face: the basic platform is completed and appropriately interfaced to an onboard microcontroller, ready for action. Now what? How does one go about transforming this creation with so much perceived potential into an intelligent entity that the very word "robot" seems to suggest? There is generally a big gap looming here between the ultimate dreams of the developer and the actual capabilities of the current prototype, probably the single most difficult hurdle to be faced.

Indeed, there have been literally hundreds of government-funded programs in robotics over the past three decades, involving teams of highly skilled and experienced researchers with budgets measured in millions of dollars. Yet despite such efforts, there still are no fielded military robotic systems (although some are getting close!), and similarly there are only a handful of commercial applications, limited in every case to very structured operating scenarios. In retrospect, the biggest stumbling block to success has been achieving a practical and reliable capability for autonomous navigation.

Approach

So how then is the everyday home hobbyist supposed to overcome this challenge? Most of us private citizens don't have easy access to massively parallel computer architectures and exotic laser-based sensors. Yet this deficiency in available resources is often the hobbyist's best asset, for necessity is the mother of invention, as the cliché goes. The home robotics enthusiast who doesn't know a particular problem is supposedly almost insurmountable will sometimes find a very practical way to get the job done, simply because he or she lacks the funds to explore some of the more expensive alternatives. Particularly if we bound the problem and don't have to formulate a 100-percent solution addressing the full spectrum of difficulties. In keeping with this spirit, this series of articles outlines a simplistic and methodical approach for incrementally bridging this technological hurdle, attaining true autonomy in an ordinary home environment.

The first step in formulating such an approach is to define the objective. It is assumed that we have already built a computer controlled platform capable of mechanical motion with measurable (at least with some degree of accuracy) displacement, speed, and direction. The novelty of such an achievement wears off quickly, however, if not augmented by some higher level of intelligence that can suitably direct the motions of the platform. Simply put, a successful mobile robot must be able to get from point A to point B, and without running into anything. To do this in a robust fashion,

without human intervention, thus becomes our objective.

A number of strategies for achieving this objective have been proposed over the years. For our purposes these can be subdivided into three basic categories: 1) reactive control, 2) representational world modeling, and 3) some combination of both. Reactive control refers to a behavior-based strategy that directly couples real-time sensory information to motor actions, without the use of intervening symbolic representations that attempt to model, in an absolute sense, the robot's operating environment. Arkin (1992) lists the following general characteristics of reactive control:

- It is typically manifested by a decomposition into primitive behaviors.
- Global representations of the robot and its surroundings are avoided.
- Sensor decoupling is preferred over sensor fusion.
- It is well suited for dynamically changing environments.

Navigation schemes that fall into the representational world modeling category traditionally take a different approach, specifically incorporating an absolute world model that represents the robot and its surroundings from a global perspective. Path planning algorithms operate on the information stored in this model to generate appropriate trajectories for robot motion. Since sensor information describing the perceived surroundings is measured relative to the robot but stored in absolute coordinates in the model, it is imperative that the robot be able to accurately determine its absolute position and orientation. Obtaining such information with the required accuracy in real-time is the fundamental difficulty with implementing the representational world modeling approach in practice.

The first article in this series describes a very basic reactive-control navigation scheme based upon a relative model, implemented about 15 years ago on ROBART I (Figure 1) as part of a crude feasibility demonstration of a robotic security concept. The use of a relative, as opposed to absolute, model eliminates the dependency on accurate and timely position information, but with some associated tradeoffs in performance, as will be discussed. A follow-on article in the Spring issue will explore a more sophisticated absolute world modeling scheme developed for use on ROBART II, an improved second-generation prototype.

Background

ROBART I was my thesis project at the Naval Postgraduate School in Monterey, CA (Everett, 1982a; 1982b). Its assigned function was to patrol a normal home environment, following either a random or preset pattern from room to room, checking for unwanted conditions such as fire, smoke, intrusion, etc. The security application was chosen because it demonstrated performance of a useful function without requiring an end-effector or vision system, significantly reducing the overall system complexity. The robot could automatically locate and connect with a free-standing recharging station when battery voltage began running low. Patrols were made at random intervals, with the majority of time spent immobile in a passive intrusion-detection mode to conserve power.

ROBART's 12V 20Ah lead-acid battery gave about six hours of continuous service and then required 12 hours of charge. Roughly one hour of power was available to locate the charging station (by means of a visual

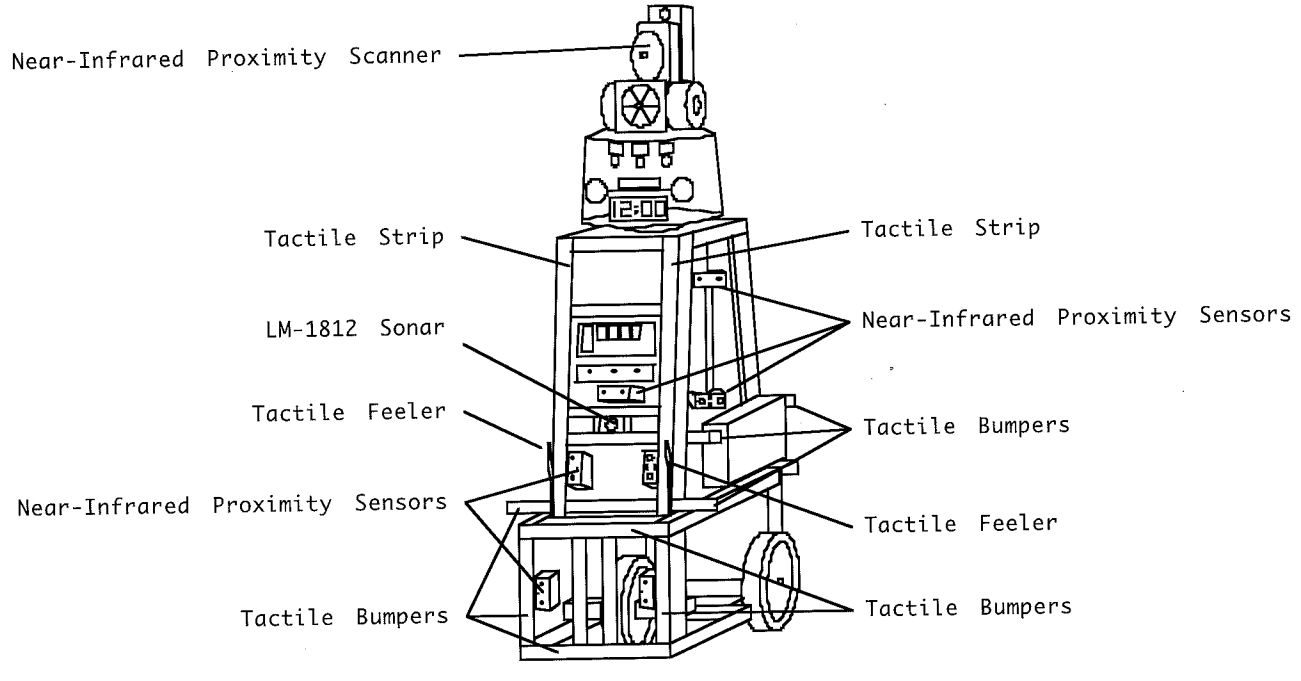

Figure 2: *Location of optical and ultrasonic collision avoidance sensors employed on ROBART I.*

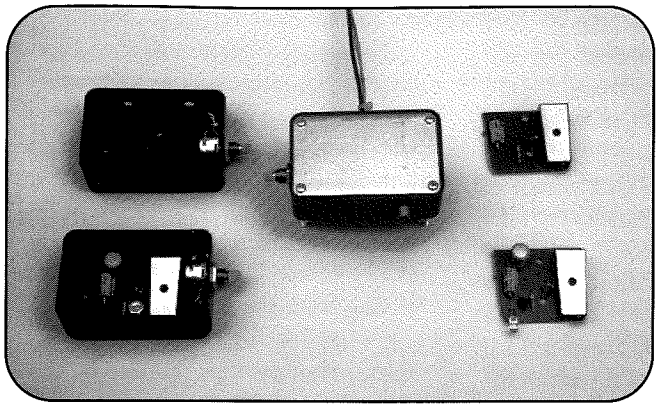

Figure 3: *The ten near-infrared proximity sensors on ROBART I were adapted from surplus circuit boards used in the manufacture of a popular toy (now obsolete). See Figure 1 and Figure 2 for placement.*

homing beacon) after the battery monitor circuit detected a low condition. The homing beacon was activated by a coded signal sent out from an RF transmitter located atop the robot's head. When a demand was sensed after connection, the recharging supply was enabled. The robot could elect to seek out the recharging station before a low-battery condition actually arose, such as between patrols.

The most simplistic reactive control capability for a mobile robot is perhaps illustrated by the basic wander routine. See the work of several research groups, including: Everett, 1982; Brooks, 1986; Arkin, 1987; Anderson & Donath, 1988. The term wander is used here to describe a behavioral primitive that involves traveling more or less in a straight line until an obstacle is encountered, altering course to avoid impact, then resuming straight-line motion. Such a capability can be simply hard-wired, rule-based, model-based, or inherent in a more sophisticated layered subsumption architecture (Brooks, 1986).

The wander routine employed on ROBART I was based on a six-level scheme of proximity and impact detection using the following sensor inputs (see Figure 2):

- A positionable near-infrared proximity scanner mounted on the head.
- A forward-looking LM-1812 sonar mounted 20 inches above the floor.
- Ten near-infrared proximity detectors (Figure 3) to sense close (< 18 inches) obstructions.
- Projecting "cat-whisker" tactile sensors to detect pending (< 6 inches) collisions.
- Contact bumpers to detect actual impact.
- Drive motor current sensors to monitor for overload condition indicative of a stall.

The first two categories were loosely classified as non-contact ranging sensors that looked out ahead of the robot for planning purposes, while the next three were considered close-in proximity and tactile sensors requiring immediate action. Drive motor overload was a last resort internal sensor in the event contact with an object was for whatever reason not detected by any of the above.

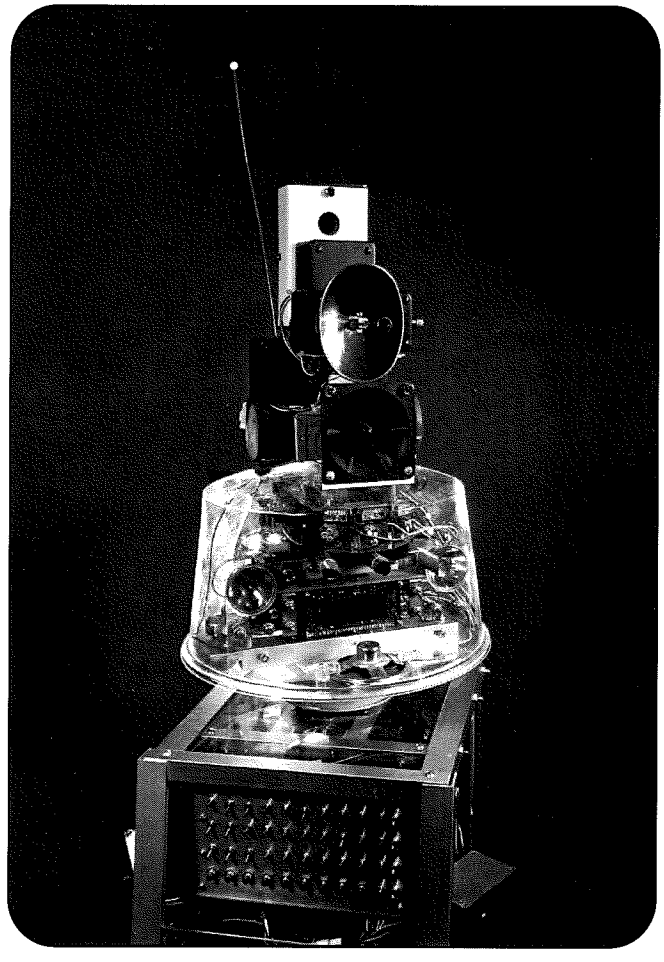

Figure 4: *The parabolic reflector at top center focused reflected energy onto a PIN photodiode detector employed in the receiver portion of the near-infrared proximity scanner.*

As ROBART I predated the introduction of the popular Polaroid ultrasonic ranging module, the forward-looking sonar was based on a Massa piezo-electric transducer interfaced to the National Semiconductor LM-1812 integrated circuit (now obsolete), originally developed for fish finders. A custom head-mounted near-infrared proximity scanner (Figure 4) provided reliable detection of diffuse wall surfaces for ranges out to six feet. This sensor could be positioned at any angle up to 100 degrees either side of centerline by panning the head, and was extremely useful in locating open doors and clear paths for travel. Excellent bearing information could be obtained–for example, both sides of an open doorway could be located within one inch at a distance of five feet.

Those sensors monitoring ROBART's close-in environment (proximity detectors, feeler probes, bumpers, and drive current overload) were considered high priority and consequently read by a maskable interrupt routine. Unless deactivated by the main program loop, the interrupt routine continuously monitored the sensor output states, and would redirect the motion of the robot in accordance with prepro-

grammed responses specifically tailored to the individual sensors in alarm. By way of illustration, a preprogrammed response for a right-front bumper impact would consist of the following steps:

- Stop all forward travel.
- Turn steering full right.
- Back up for x number of seconds.
- Monitor rear bumper for impact.
- Stop and center steering.
- Resume forward travel.

To facilitate somewhat more intelligent movement than the purely reactionary "bump-and-recover" interrupt routines, the intermediate-level software would repeatedly poll the sonar and near-infrared scanner on each pass through the main loop. These longer range sensors were tasked with monitoring the area in front of the robot and storing a suitable representation of detected targets in a relative model as illustrated in Figure 5. The wander algorithm reacted to the information in the model by choosing the least obstructed direction for continued transit in the event the forward path became blocked. Since all zones were equally weighted in a binary fashion (either blocked or clear), the least obstructed direction was taken to be that opening defined by the largest number of adjacent clear zones. The inherent simplicity of this modeling scheme enabled on-the-fly collision avoidance response, without the robot having to "stop and think" before continuing on its way.

Hallway Navigation

The role of a world model (for our purposes) is essentially two-fold: 1) to represent the relative locations of surrounding obstructions in order to formulate an appropriate avoidance maneuver and 2) to support the global generation of a path to the destination goal. Implementation of this second attribute was addressed by adding a general awareness of the relative locations of the various rooms along either side of a definitive hallway, resulting in a fairly robust navigation capability that lent itself to implementation on an 8-bit microprocessor. The recharging station optical beacon was suitably positioned in a known location as shown in Figure 6 to assist the robot in entering the hallway. Once in the hallway, the robot would move parallel to the walls in a reflexive fashion, guided by the near-infrared proximity sensors. General orientation in the hallway could be determined by knowing which direction afforded a view of the beacon.

With a priori knowledge of where the individual rooms were situated with respect to this hallway, ROBART I could proceed in a semi-intelligent fashion to any given room, simply by counting off the correct number of open doorways on the appropriate side of the hall. Each room was assigned a number, with odd-numbered rooms on the left in increasing order when moving away from the beacon (see again Figure 6). A dedicated behavioral routine took care of the specifics associated with guiding the robot through the detected door opening into the room of interest. Once inside a particular room, the robot's wander routine would tend to hug the wall, traversing a rectangular path which eventually brought it back to the open doorway. By again invoking a doorway penetration behavior, the robot could re-enter the hallway and turn in the direction of the next room to be visited.

ROBART I thus knew in which room it was located at any given time, but not its absolute X-Y position or heading within that room. In a global sense, this could be regarded as operating from a relative navigational map as opposed to an absolute map depicting the actual floorplan. Interestingly enough, we humans tend to function in much the same fashion, in that we know where objects are with respect to one another, and have but a fuzzy feel for their absolute representation. Upon entering our darkened kitchen for a late-night snack, for example, we reach instinctively to a certain location for the light switch, relative to the doorway of entry. Such a relative representation makes it much easier to retain the necessary information in memory for a large number of locations. Imagine how cluttered our brains would get if we were forced to store an absolute model of everything we observed. The same principle applies to an 8-bit microprocessor with a limited 64-Kb address space.

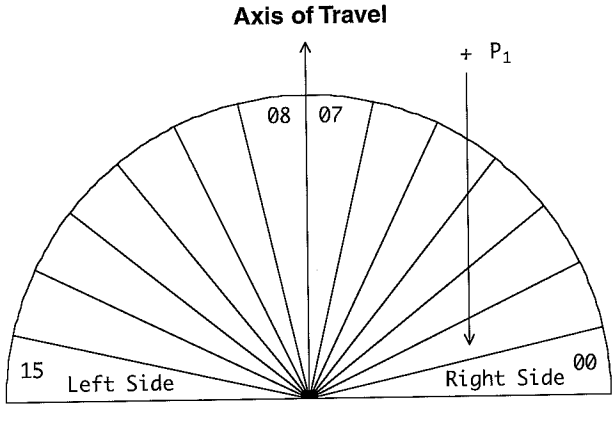

Figure 5: *The sensor fusion model employed on ROBART I consisted of sixteen wedge-shaped zones relative to the direction of travel (Everett, 1982b).*

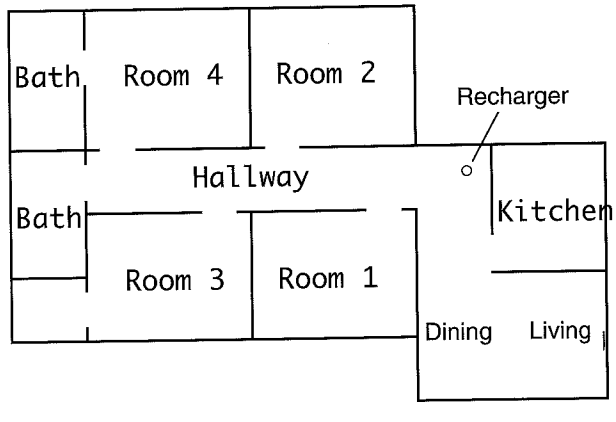

Figure 6: *Floorplan of the operating environment for ROBART I showing the location of the recharging station at the right end of the hallway.*

Requirements

To replicate a similar navigational capability on a mobile platform of your own design, assuming the floorplan reflects a suitable hallway oriented layout, only a few simple behavioral primitives must be addressed:

- Wander–Causes the platform to move forward, turning away from any detected obstructions in its path.
- Find Door–Locates an open doorway on the specified side of the robot.
- Enter Door–Reflexively guides the robot safely through the detected doorway.
- Verify Direction–Confirms the robot is traveling in the proper direction (optional).

In addition, some means of establishing the relative relationships of the rooms is required, such as a linked-list representation, or even more simply, a lookup-table as was implemented on ROBART I. A couple of state variables are also required to tell the robot which room it currently occupies, and in the case of the hallway, the direction of motion (in terms of increasing or decreasing room numbers, see Table 1). The next section discusses the actual implementation of these basic behavior primitives, followed by one possible approach to developing a simple relative world model for hallway navigation.

Implementation

The collision avoidance sensing needs of a simple wander routine can be met with three optical proximity sensors arranged in a fan-shaped pattern as depicted in Figure 7. The Banner Engineering *Valu-Beam® SM912D* diffuse proximity sensor (Banner, 1993a, 1993b) shown in Figure 8 performs well in this mode, draws only 40mA at 12V DC, and provides a simple binary (target or no target) output line for ease of interface. A gain potentiometer adjustment is provided to set the maximum effective detection range for typical diffuse target surfaces (such as a wall) anywhere out to about eight feet. Detection setpoints of 12 inches for the left and right sensors and 18 inches for the center sensors are good starting points, but you should experiment for best results. The cost of this unit is a bit high by most hobbyists' standards, about $80, but the payback is immediate availability and very reliable operation. Jones and Flynn (1993) describe a method for making your own low-cost proximity sensor with a somewhat shorter effective range, based on the Sharp GP1U52X near-infrared detector module. Similarly, McManis (1995) presents details for building a multi-zone proximity sensor using the Sharp IS1U60 detector.

Alternatively, three ultrasonic transducers can be multiplexed to a single Polaroid ranging module (Everett, 1995a) to achieve similar effect, but the interface requirements are a little more complex. The highest probability of obstacle detection is of course achieved with redundant coverage from both optical and ultrasonic sensors. For such dual-mode implementations, the polling software can simply logically OR the outputs together for any sensors aligned along a common axis. In this fashion, if either a sonar or an optical sensor reports an obstruction in a particular direction, the robot will turn away. For sake of simplicity, however, the remainder of this discussion will be limited to a basic configuration involving three optical proximity sensors only.

The simplest implementation of the wander algorithm is probably rule-based (as opposed to the model-based collision avoidance strategy depicted in Figure 5), taking the form of a series of conditionals that alter the robot's forward travel in response to potential obstacles detected by the forward looking sensors. Obviously, if the right-hand sensor sees a target, the robot turns left, whereas if the left-hand sensor sees something, the robot turns right. If the center sensor also sees the target, the rate of turn is increased. However, if only the center sensor sees a target, the robot should turn either left or right in accordance with a preset variable

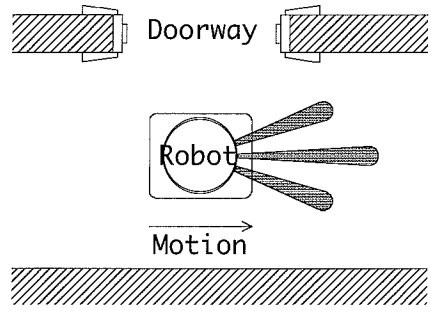

Figure 7: *Three optical proximity sensors, three sonar sensors, or the superposition of both can be used to implement a simple wander routine. Maximum effective detection range for the center unit should be just a bit further than for left and right.*

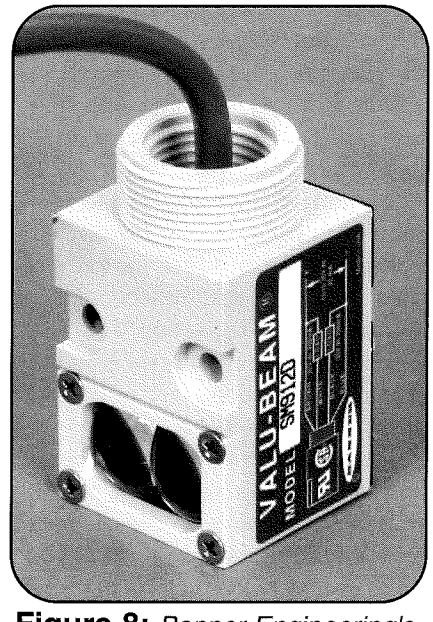

Figure 8: *Banner Engineering's* Valu-Beam® SM912D *diffuse proximity sensor has an adjustable gain control that sets the maximum detection range to a diffuse reflector such as a wall surface. Unpainted sheetrock can be detected at a distance of eight feet.*

Value	Meaning
00	robot not in hallway
01	decreasing room numbers
02	increasing room numbers

Table 1: *Interpretation of possible values for the state variable representing the robot's direction of motion in the hallway.*

(turn_preference). And finally, when all three sensors detect an obstacle, the robot should pivot in place (in the case of a differentially steered platform) or back up slightly before turning (in the case of tricycle or Ackerman steering).

To ensure stability, these predefined response actions should continue for some specified time interval (0.5 to 2 seconds typical) after the condition clears. When sensor gain, turning rate, and delay parameters are appropriately tuned through experimentation, this limited amount of hardware can be used to implement an effective wall hugging motion that guides the robot safely down the hallway. Similarly, prespecifying a pivot direction for the blocked condition causes the platform to traverse the perimeter of a room in a rectangular pattern, either clockwise or counterclockwise, depending on the current value of turn_preference.

The find door behavior requires some physical means for locating an opening on either side of the hallway. This task can be readily accomplished with a pair of side-looking *ValuBeam® SM912D* sensors set for a maximum detection range of about four feet. Alternatively, two side-mounted sonar transducers can be employed for this purpose. Optical sensors have a bit of an advantage in that they can be aimed forward just a bit to give advance warning in time to turn, whereas sonar sensors work best when the beam is kept nearly orthogonal to the target surface, due to problems associated with specular reflection (Everett, 1995a). If your robot has a positionable head, you can get by with a single sensor and mechanically point it in the desired direction. This is the approach employed on ROBART I, but not recommended unless the positionable head is required for other purposes as well. It is much simpler to electronically multiplex two (or more) sensors.

The enter door primitive requires no additional sensor hardware. Once the doorway is detected by find door, a turn is initiated in the proper direction. A slight delay might be required first to achieve the proper starting point, but this can easily be determined through trial and error. The actual rate of turn must also be optimized for the dynamics of your particular platform. The secret to success here lies in the fact that the wander behavior is not inhibited during execution of the enter door routine. If at any time the forward looking sensors that support wander detect a target, the enter door routine is terminated and wander takes over. The reflexive avoidance behavior of wander then adjusts the platform's direction of motion accordingly to keep it from impacting the sides of the doorway, the door itself, or an adjoining wall. The result is seamless and deceptively intelligent continuous motion.

High-level software determines which way the robot should turn when re-entering the hallway by checking to see if the current room number is odd or even, comparing the next room number on the visit list to the current room number, and then setting the turn_preference register accordingly. The bigger challenge is deciding when to make the turn. If no doorways directly oppose each other in the hallway, the solution is rather trivial, as the opposite wall (Figure 9) will force the wander behavior to turn at a distance of approximately 18 inches when the center proximity detector picks up a reflection. The calculated value stored in the turn_preference register ensures the robot turns in the correct direction.

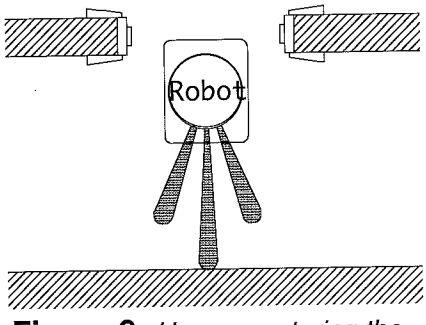

Figure 9: *Upon re-entering the hallway, the robot turns in the precalculated direction stored in turn_preference when the center proximity sensor detects the opposite wall.*

If there is another open doorway directly across the hall this quick and dirty solution fails for obvious reasons. In this more complicated scenario, you must use the two side-looking proximity detectors to determine when the robot has passed through the doorway and into the hall, and then force a turn. You can postpone solving this dilemma for the time being, if so desired, by simply closing off one of the two doors and eliminating that room number from the linked list representation, to be discussed later. The idea here is to start simple and then add smarter routines as you become more experienced. You can learn a lot about sensor interaction with various target surfaces by observing your initial prototype, and this valuable experience will provide significant insight into how to improve your design.

The verify direction primitive allows confirmation that the robot is indeed properly oriented in the hallway (moving in the direction of increasing or decreasing room numbers). While not absolutely necessary, this feature adds significantly to the robustness of the relative navigation scheme. On ROBART I this task was accomplished by activating the optical beacon on the recharging station and then checking to see if the robot could detect it–allowing the robot to determine if it was facing the beacon. This technique was available at no extra cost since the beacon tracking hardware was already in place to support automatic recharging, but such may not be the case with your design. In any event, a much simpler and more effective solution is now possible due to the recent introduction of low-cost fluxgate compasses. While the more expensive models are accurate to within about a degree, even the very cheapest versions (around $40 at many auto parts stores) are more than sufficient to eliminate a 180-degree ambiguity in heading. Chapter 12 in *Sensors for Mobile Robots* (Everett, 1995b) describes the principle of operation and interface requirements for these devices in detail.

The most expeditious representation of the room inter-relationships is probably in the form of an indexed-array data structure as illustrated in Table 2, which can be easily implemented on an 8-bit microprocessor using the indexed addressing mode. The index serves as a pointer to the current room number, and by decrementing or incrementing the index accordingly, you can step through memory in the same order the robot would encounter the rooms as it traverses down the hall. Checking to see if the destination room number is odd or even determines on which side of the hall the doorway will be found. Upon first glance it may seem like the array structure is not even necessary, as the data value merely echoes the index value, and a simple variable current_room would suffice. For an ideal scenario such as depicted in Figure 6, that is indeed correct. The use of an array, however, allows additional information about each room to be encoded in the upper nibble, while the lower nibble represents the room ID number.

For example, the simplistic case illustrated above assumes there are an equal number of rooms on both sides of the hall, and repetitive increments of the index will step through memory in precise correlation to the robot's passage by each of the rooms. In reality, such may not be the case, as rooms can vary in size. An easy way to preserve the validity of the algorithm in this situation is to allow for the representation of phantom rooms that subdivide the space occupied by a single large room into the appropriate number n of smaller rooms. Since n-1 of the phantom rooms will not have detectable doorways, they are specially marked by setting a designated bit in the upper nibble so the algorithm will increment the index without waiting for actual doorway detection. Other bits in the upper nibble can be used to mark rooms that have doorways facing another door directly across the hall, rooms containing battery recharging facilities, or even rooms which lead to other rooms.

The basic algorithm (not yet enhanced to accommodate phantom rooms) is presented in flowchart format in Figure 10. If the robot is exiting room 1 and wants to go next to room 4 (see again Figure 6), a left turn is in order upon entering the hallway, since the current room is odd and 4 > 1. The index is initially pointing to room 1. The algorithm first resets the direction_of_motion flag, then waits until the robot is traversing down the hallway with a wall detected on both sides (in order to avoid mistaking the doorway just exited for the anticipated detection of room 2). After this hallway verification condition is achieved, the right-hand doorway detector should next pick up the opening that marks the entrance to room 2. The room index is incremented but does not yet match the destination room number, so the software loops back up and waits for doorway 2 to clear. After walls are again detected on both sides of the robot, the left-hand detector is monitored for the opening at room 3, and so forth. When the room index finally matches the destination room number, the algorithm calls the enter_door routine and turns the robot left or right, based on whether the room to be entered is odd or even, and the current direction of motion.

Table 2: *A one-dimensional array data structure contains all the necessary information to describe the room inter-relationships in support of the relative navigation scheme. The upper nibble of the data value can be used to encode special information, while the lower nibble reflects the assigned room number.*

Address	Value
XX00	00
XX01	01
XX02	02
XX03	03
XX04	04
XX05	05
XX06	06

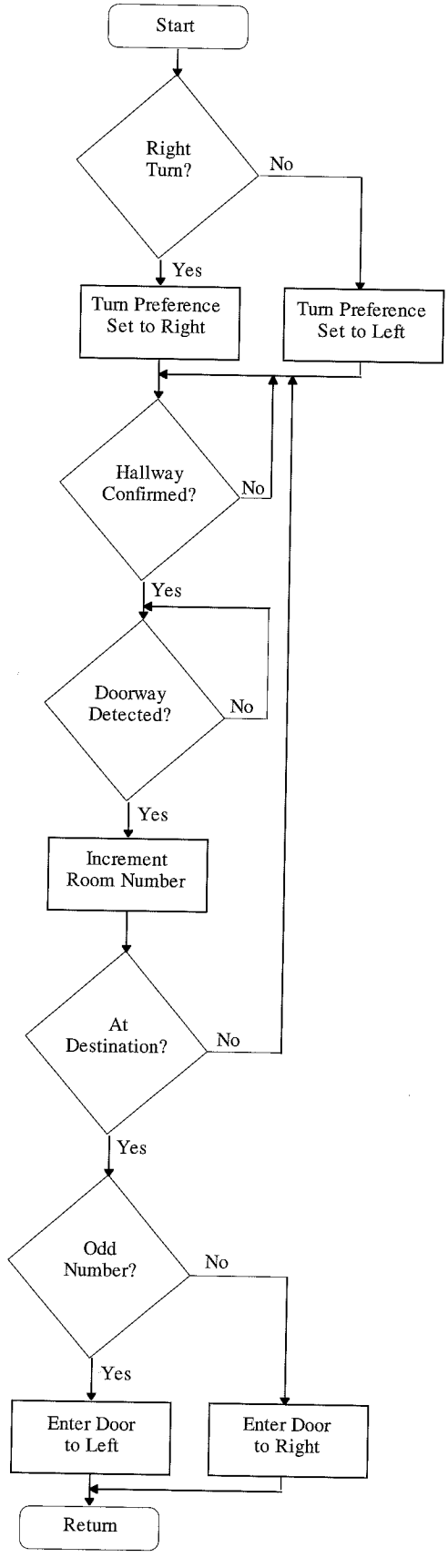

Figure 10: *Flowchart of the basic hallway navigation algorithm.*

Helpful Hints

The single most important rule is to start simple and not try to solve all possible problems with your first design. You don't, for example, have to be able to navigate in every room in your house right off the drawing board. Select a couple of representative rooms along your hallway that lend themselves well to the concept, and get the system up and running. Watch what happens and go from there. Once you have your control algorithms stabilized through optimal turn rates, velocity profiles, and sensor gain settings, you can start adding to the robot's overall intelligence through incorporation of smarter algorithms. Observe the inevitable failure modes and determine what made the system do what it did, then adjust accordingly.

The left and right collision avoidance sensors should be angled out just enough to pick up the wall surface when the robot gets to within about four to six inches, running parallel to the wall. If the fan-out angle is too large, the robot may zigzag instead of hugging the wall, and there may also be some blind spots in the forward collision avoidance coverage. An angle of about 20 degrees from center should work fairly well. Zigzag motion will also result if the avoidance reaction is too severe, due either to excessive turning rate or too much execution time.

The maximum range of the doorway detection sensor should be set at a greater distance than will be observed under worst-case conditions in the hallway, yet not enough to pick up a distant target inside the room. Four feet is a recommended starting point for a three-foot hallway. You may want to increase it a bit if you plan to scan the sensor back and forth in search of a doorway from the inside of a room. Don't set the gain high enough to pick up the far wall in the hallway, though, or you'll never find the door.

It's best to "debounce" your doorway detection sensor outputs through low-pass filtering to eliminate erroneous readings. In other words, require the sensor to show an opening for some number of consecutive reads, with a slight delay each pass through the loop, before reacting to the data. This way if a momentary loss of target occurs due to specular reflection, the robot won't turn prematurely. When an actual doorway is present, the sensor output will toggle and remain in the new state for an appreciable length of time. Don't delay too long, however, or you'll overshoot the door.

Note that room doors usually swing inward, and almost always are situated in a corner so the door folds back 90 degrees against an adjoining wall (Figure 11). This arrangement allows for more efficient use of wallspace and makes the door easier for humans to shut upon exiting the room, relative to a door that folds back 180 degrees against its own wall. Such a structured relationship can be exploited to a certain extent by the robot when attempting to re-enter the hallway, in that the door itself can be used to trigger a turn through the open doorway. For example, if the robot were to enter Room 1 in Figure 6 and make three right turns as it followed the wall perimeter, it would then be approaching the open doorway as shown in Figure 11. By resetting turn_preference at that point to left, the robot would exit through the open doorway back into the hall.

In the real world, however, most rooms are not clutter free as implied above. If you have problems locating the doorway in order to exit a room, you may want to experiment with polarized retro-reflective optical sensors, such as the Banner Q85VR3LP. The advantage of this type over the diffuse variety is unambiguous detection of a retro-reflective tape that can be attached to doorway edges for positive identification. This type of sensor will not respond to diffuse target surfaces such as walls, or even direct reflection of its beam from a mirror. A retro-reflective target shifts the beam polarization in a unique fashion, and only this returning polarization pattern will trigger the detector. The disadvantage of this scheme of course is that it requires some modification of the robot's operating environment, although the tape strips are relatively unobtrusive.

Once you get the basic behavior primitives up and running, the fun really starts! Buy a cheap electronic compass, interface it to your onboard computer, and see what performance improvements result from this additional navigational information. The task of exiting a room, for example, is greatly simplified if you can positively identify the correct wall to scan for an opening. Implement a rudimentary dead-reckoning capability and improve your algorithms even more. Incorporate time-out or exit conditions for your behavior primitives to preclude getting caught in an endless loop when something goes wrong. Augment your code with trap recovery routines that take over if the robot gets boxed into a repetitive cycle that leads nowhere. Add exception handlers to deal with opposing doorways, a door situated at the very end of a hallway, or a room with two doors.

Perhaps you may choose to incorporate a look-ahead scanner of some type and expand your world model to include a forward-area representation similar to that depicted in Figure 5. You can even develop a graphical map display that shows which room the robot has most recently detected. Take it one step at a time and learn as you go. Mobile robotics is probably the ultimate hobby from the standpoint that even a well-planned project is never finished, and the continuing satisfaction of making further improvements is essentially unlimited.

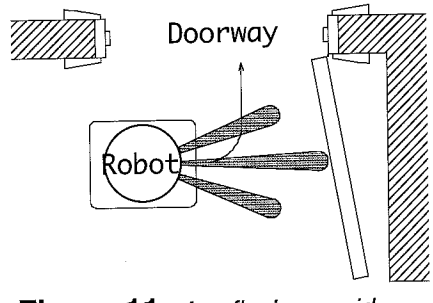

Figure 11: *A reflexive avoidance maneuver (preset in this case for a left turn) triggered by detection of the open door can be exploited to facilitate room exit through the adjacent doorway.*

Summary

This first article discusses a novel low-cost solution (admittedly over simplistic) to the global navigation problem that can be easily implemented on an 8-bit microprocessor at minimal cost. The approach is basically insensitive to the type of drive and steering configuration employed on the platform, and does not even require a dead-reckoning capability of any sort. We have bounded the problem by limiting our operation to indoor environments with easily traversible floor surfaces, and more specifically, solely within the confines of a series of rooms arranged in ordered fashion along either side of a defined hallway. Quite obviously these limitations preclude any meaningful consideration of such a relative navigational scheme in serious real-world applications, but it is hoped this concept will provide an educational stepping stone for the interested reader with somewhat limited financial resources and/or programming experience. A number of suggestions were included for enhancing this most basic implementation for those interested in probing further. An even more sophisticated navigational solution based on representational world modeling will be discussed in the next issue.

About the Author

Commander H. R. (Bart) Everett, USN (Ret.), is the former Director of the Office of Robotics and Autonomous Systems at the Naval Sea Systems Command, Washington, DC. He currently serves as Technical Director for the tri-service Mobile Detection Assessment Response System (MDARS) robotic security program under development at the Naval Command Control and Ocean Surveillance Center, San Diego, CA. Active in the field of robotics research for over 20 years, with personal involvement in the development of 11 mobile systems, he has more than 70 technical papers and reports published and 16 related patents issued or pending. He serves on the Editorial Board for Robotics and Autonomous Systems magazine and on the Board of Directors for the International Service Robot Association, and is a member of Sigma Xi, the Institute of Electrical and Electronics Engineers (IEEE), and the Association for Unmanned Vehicle Systems International (AUVSI).

everett@nosc.mil

http://www.nosc.mil:80/robots/index.html

Portions of this article were adapted from the book *Sensors for Mobile Robots: Theory and Application*, published by AK Peters, Ltd., Wellesley, MA, ISBN 1-56881-048-2.

References

Anderson, T.L., Donath, M., "Synthesis of Reflexive Behavior for a Mobile Robot Based Upon a Stimulus-Response Paradigm," SPIE Mobile Robots III, Vol 1007, W. Wolfe, Ed., Cambridge, MA, pp. 198-211, November, 1988.

Arkin, R.C., "Motor-Schema-Based Navigation for a Mobile Robot: An Approach to Programming by Behavior," IEEE International Conference on Robotics and Automation, Raleigh, NC, 1987.

Arkin, R.C., "Behavior-Based Robot Navigation for Extended Domains," *Adaptive Behavior*, Vol. 1, No. 2, MIT, Cambridge, MA, pp. 201-225, 1992.

Banner, *Photoelectric Controls*, Product Catalog, Banner Engineering Corp., Minneapolis, MN, 1993a.

Banner, *Handbook of Photoelectric Sensing*, Banner Engineering Corp., Minneapolis, MN, 1993b.

Borenstein, J., Koren, Y., "Real-Time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments," IEEE International Conference on Robotics and Automation, Vol. CH2876-1, Cincinnati, OH, pp. 572-577, May, 1990a.

Borenstein, J., Koren, Y., "Real-Time Map Building for Fast Mobile Robot Obstacle Avoidance," SPIE Vol. 1388, Mobile Robots V, Boston, MA, November, 1990b.

Brooks, R.A., "A Robust Layered Control System for a Mobile Robot," *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 1, pp. 14-20, 1986.

Everett, H.R., "A Computer Controlled Sentry Robot," *Robotics Age*, March/April, 1982a.

Everett, H.R., "A Microprocessor Controlled Autonomous Sentry Robot", Masters Thesis, Naval Postgraduate School, Monterey, CA, October, 1982b.

Everett, H.R., "Understanding Ultrasonic Ranging Sensors," *The Robotics Practitioner*, Fall, 1995a.

Everett, H.R., *Sensors for Mobile Robots: Theory and Application*, ISBN 1-56881-048-2, AK Peters, Ltd, Wellesley, MA, 1995b.

Jones, J., Flynn, A.M., *Mobile Robots: Inspiration to Implementation*, ISBN 1-56881-011-3, AK Peters, Ltd., Wellesley, MA, pp. 106-111, 1993.

McManis, C., "Turning Toys into Tools," *Circuit Cellar INK*, Issue #63, pp. 24-35, October, 1995.