Autonomous Robot Independent Study Project

Larry Grove

CSS498

July 2004

Table of Contents

Objectives

Autonomous robotics is an area of academic and commercial research as well as popular interest. It holds interest because of its interdisciplinary melding of mechanics, sensors, computers and artificial intelligence.

To learn more about the subject, I developed a robot for an Independent Study course at the University of Washington, Bothell. The goal of the class was to learn more about autonomous robotics. The goal for my robot was to move around an office-type environment, avoiding obstacles and mapping its environment.

Historical overview

Popular science-fiction literature has long been a catalyst for thought in the sciences. In robotics, this has been no different. Writers (like Isaac Asimov) were significant early creators of robot portrayals. And their books influenced the direction of early robotics research work.

At first in the field called Cybernetics, biological entities were studied as if they were machines and intelligence was modeled as a control-theory problem.

Later when the computer became a capable tool, complex robotic world models were developed. The ultimate goal was generally an analytical human-like intelligence since that was the highest standard of comparison. However, it became evident that this was an extremely difficult challenge. In addition, it took much time to process the intelligence models. Eventually many deliberative knowledge-based systems failed because they could not stay synchronized to the real world in quick-response situations.

In response, alternative schools of thought emphasizing behavior-based intelligence developed. Commonly, sensors directly actuated the behaviors. With an ingenious set of sensors and behaviors, a robot could successfully interact by instinct in an unstructured environment. However, like the insects that inspired them, they retained no significant memory of their environment.

Subsequent hybrid designs pulled advantages from both these schools of thought. They have layered architectures: Behaviors driven by sensors enable the robot to live in the moment. But they also have knowledge modeling that enables planning and long-term behaviors.

Research literature

I had followed the popular robot culture for some time, primarily through the Internet. I also purchased the excellent Mobile Robots book by Jones and Flynn (1993) to study simple behavior-based robots.

The Internet also enables access to scholarly resources and publications of a number of University robotics departments. From these sites I studied both historic and current papers on robotics issues. Significant sites include MIT, Carnegie Mellon, Stanford & the University of Washington.

The paper, A Robust Layered Control System for a Mobile Robot (Brooks, 1985) introduced behavior-based robotics intelligence and a subsumption architecture to implement behavior control. Jones and Flynn worked and studied with Brooks. Their Mobile Robots book continued this theme in the context of implementation.

Arkin's book Behavior-Based Robotics (1998) expands this topic with consideration to both Brook's minimalist behavior-based intelligence and the traditional deliberative cognitive intelligence. Topics covered include architectures, representation systems, perception issues, and adaptive techniques.

Robotic Mapping: A Survey (Thrun, 2002) reviews the problems involved in mapping and covers eight major algorithms used to perform mapping and their strengths and weaknesses. Because of sensor noise and uncertainty in the mapping process, virtually all mapping techniques use probabilistic methods. Hard problems in the topic include -how systematic errors in the information gathering process may be accommodated, -how to manage the quantity of data needed to represent the map, -how to recognize a previously mapped place and merge data taken there at different times, -how to accommodate changes of an environment over time, -and how to choose effective paths within a map.

Mapping as a whole is one of the most difficult problems in mobile robotics. The most successful solutions are often very complex. Thrun (Stanford) & Fox (UW) are primarily focused on this area. Most of their publications are highly technical, about specific mapping approaches and well beyond the scope of this project. Nevertheless, a survey of their papers was useful to understand the state of the art in the field.

Military researchers and open source projects also proved to be valuable research resources. Varveropoulos (2000) described a practical implementation of robotic mapping that was very useful. Lucas (2000) provided a mathematical model of differential steering systems.

Sensors are critical for successful operation in real environments. Everett's book Sensors for Mobile Robots (1995) is a comprehensive survey of sensors with emphasis on non-vision types. It is an excellent reference for serious robotisists. Everett's career was spent in the Robotics group of the Space and Naval Warfare Systems Center,

Advanced hobbyists and commercial providers also provided useful practical information. Part manufacturers often had a good selection of application notes for their products at their web sites.

Approach

Personal goals for this project were that it contain several design elements: It would have a simple hybrid intelligence with both reactive behavior functions and elements of a deliberative knowledge model. It would have distributed processing. I wanted to include microcontrollers (MCUs). It would work in a real world setting, but structure and environment simplification would be OK.

The basic design partition was intended to leverage simplicity and still be expandable to handle unforeseen problems. I would use a differential (wheelchair) drive on a low flat platform. This type of locomotion has the advantage of very simple mechanics, and great flexibility of movement (it can turn in place).

Individual MCUs would provide interfacing to the real world. They would communicate with each other and a main controller by serial interface. The main controller would be a laptop sitting on the platform.

Component choices

Even though my locomotion model was simple, I had a hard time deciding on a simple yet durable implementation. Eventually I decided that main wheels would come from in-line skates and be driven by the mechanical portion of RC hobby servos. For a third point of balance I used a universal-direction wheel as a castor.

Since the robot was to perform mapping it would need range-finding sensors. Sharp has a line of IR (infra-red) optical triangulation sensors with a good reputation. Their output is a DC voltage, but it is not proportional to distance. I would have an MCU digitize the value and the laptop perform the linearization function.

I wanted completely contained MCU devices. The PIC processor line and some members of Motorola's 'HC08 line fulfilled this requirement. I had worked with Motorola 8-bit processors before and liked their flat-memory architecture, stack orientation and instruction set. The PICs seemed to have architecture quirks that I didn't feel comfortable with, so in the end I chose the an 'HC08.

The 68HC908KX8 unit has 8K flash, 198 bytes RAM, Pulse Width Modulation (PWM) control, timers, analog to digital converter (ADC), RS-232 serial, on-chip clock generation and many other features. Once programmed it only needs power to be useful. It is a 16 pin IC, available in a standard through-hole DIP package. The 'HC08 units have built-in breakpoint hardware and ROM that supports a debug mode. Two pins are needed for chip power, and the debugger needs as few as three more pins. There are a few other details, but basically that leaves 11 pins for I/O.

Metrowerk's industrial-strength Codewarrior Integrated Development Environment (IDE) is available for free use with modest code size license limits. It supports C, C++ & assembly and has an integrated MCU simulator and debugger.

The IDE comes with tools that can create drivers for the on-chip peripherals. The abstraction could be useful for someone familiar with the chip capabilities. However, I found it more productive to write drivers myself in assembly because it enhanced learning and then I knew what the code was doing. Initially, because I was comfortable in assembly, I thought I would do the whole project in it. However, later when I needed to do more than single-byte arithmetic, I found the C to be valuable.

A related point is that I didn't want to buy a processor board with a turnkey robotic control environment because I wanted the practice of software development closer to scratch. The self-contained execution environment of these chips makes hand-built circuits possible. And these chips may be useful for other types of projects in the future.

Initially I thought to use another member of the 'HC08 family for the project. However after I had invested some time into it I found that it did not have hardware support for serial communications. Software could do the job, but I didn't want to dedicate that much computing budget to serial. I was very glad to find the 68HC908KX8 that has serial in hardware.

Physical construction

I found an aluminum shroud from some junked piece of equipment. The aluminum was easy to work, light and strong. Many robots that I had seen in my research looked fragile. I built mine so that it could carry some weight (a laptop computer).

Although a solderless breadboard was useful for initial prototyping, something more durable was needed for the robot. After hand-drawing the circuits, I used a combination of wirewrap and point-to-point construction.

The battery pack I used puts out about 7.5 volts. The motor driver uses this voltage directly. However, each circuit board has a voltage regulator for the 5 volts required by the logic circuits. The regulators enable noise isolation between circuits. Because the frame is conductive aluminum, I used the technique used in cars and made the body be the negative power conductor.

Motion controller

The ‘HC08KX8 chip has 2 PWM outputs, so I was able to control two motors with one MCU. The MCU controls an H-bridge driver circuit that enables the motor to be reversed while being supplied from a single voltage.

Range sensor

The optical range sensors pulse their illumination LED with a fairly high current for increased range and signal noise immunity. This can produce significant electrical noise spikes. Attention to power (especially grounding) and modest capacitive filtering can control this.

For the range finders I was interested in range information in a full circle around the robot. I could sweep in a circle with either the robot itself, or I could use a separate sweep mechanism. Since I was already concerned with accuracy of robot positioning, I didn't want it to move unnecessarily. Another consideration was that the finders have a slow 40 ms update rate, so I couldn't scan them around too fast. My solution was to use a set of rangefinders in an X configuration mounted on an RC servo. They would be swept back and forth. Paralleling the four increases the rate of data acquisition.

Raising the range finders above the robot platform avoids noise from floor reflections.

In this case the 'HC08KX8 chip does all the interfacing: generation of the servo pulse train (for sensor scanning) and digitization of the sensor data. Power and data (serial) are the only connections to the module.

MCU programming

Programming the MCU on-chip peripherals went pretty well. There was a fair amount common code between the MCUs (startup, RTC, serial I/O and parsing, etc). Two things did take more time than I expected: -the velocity motion control algorithm development (first half done in assembly, then redone in C) with its debugging, -and the serial data parsing and handling.

Because the MCUs are daisy-chained, they have to pass data not meant for them and respond to data that is addressed to them. This requires bulletproof code that does not fail under constant data flow at the same time the MCU additionally performs its sensor/motion interfacing tasks. Since each MCU could add data to the serial stream they buffer both the incoming and outgoing characters. The buffer (of 20 bytes each) is larger than any single command-response set.

Each MCU has a simple command and reporting protocol. It looks at each incoming ASCII word (delimited by whitespace). Commands are in lower case. The first character is the MCU address. The second character is the function command. Command arguments follow when needed. Recognized commands are echoed verbatim (because multiple MCUs might need to respond). The MCU also gives an upper case response/report.

Motion control algorithms

Motion control needs to be closed-loop for this robot because straight-line movement requires the drive wheels to have matched velocity. I followed the technique used in Jones & Flynn, which is velocity-oriented. It works, but control is a bit sloppy. It would not have been good enough for mapping except that the robot has an optical encoder on each wheel. The motion controller counts encoder lines, so I do know robot position. It is likely that a position-oriented control algorithm would be better for the robot.

The encoders have a resolution of 40 lines / revolution. The motors produce 2 revs / sec at full speed. Since velocity is determined from line_count / unit_time, I would have difficulty getting good velocity resolution at low speeds and/or high update rates. To help things I used a technique that measures the period of each line count. There were no dedicated hardware resources in the MCU to do this, so it had to be performed by polling. Since it would be most convenient to calculate the velocity at regular intervals, the software counts the line events and sums the total period for those events. The final velocity calculation is line_count / total_period.

The motion control method in Jones & Flynn uses techniques from classic Proportional-Integral-Derivative (PID) control. A velocity is commanded to the software and a velocity is measured. The power sent to the motors (by PWM control) is in direct proportion to the error between the set and measured values. The measured velocity difference (error) between the two is accumulated (integrated). A proportion of this second error is also added to the motor power values so that ongoing difference between the velocities of the two motors is continually forced to zero.

A significant improvement to the motion system would be to increase the encoder lines / rev. At present it is 40. If it were (for instance) 1000 counts / rev, then I could increase the update rate for tighter control and still have better resolution. However, it may be physically difficult to achieve a higher encoder resolution.

In operation, the velocity for each motor is set by remote command. A command also retrieves the current position count of the motors.

Range sensor algorithms

The range sensor software is somewhat simple. Interrupt control continuously advances the pulse width servo position command with an auto-reverse function so that it continuously scans. The scan speed and range are set by remote command. The interrupt control also continuously digitizes the range sensors. A command retrieves the current position of the servo and the values of the range sensors.

Main control program

I wrote the main controller software in a language called LabVIEW (from National Instruments, Austin, TX). It is a graphical language that is very effective for test and measurement applications. Writing code in LabVIEW is much like drawing an electronic circuit diagram. In fact, I think that people with a hardware background feel more immediately comfortable with it. Its graphical nature also works well with graphical output.

I started development with a proof-of-concept map of a short hallway in our house. This was a simple Boolean occupancy grid. Each marked dot was an endpoint of an optical ranger vector. There was no statistical processing. In Figure 1, each dot is a square centimeter and the robot only moved in a straight line.

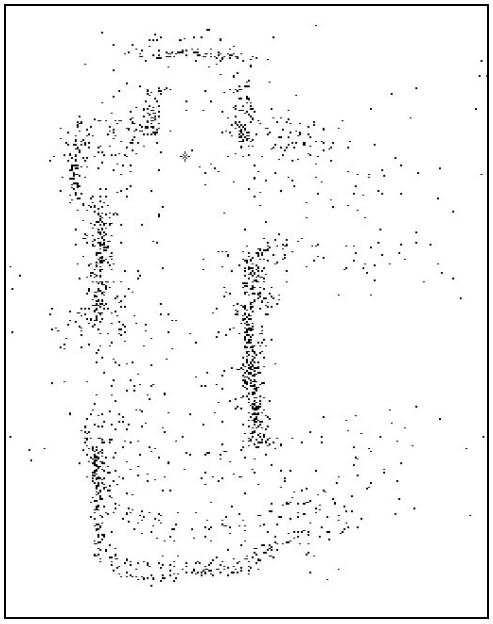
Figure 1

In Figure 1, it appears that the shorter-distance rangers max out at about a meter. Even if they notice something 3 meters away, it gets reported as a meter. Because of this I think the curved arcs down the middle of the hallway are false readings. Therefore probability curves will be needed to assign confidence of a measurement's reliability with cut-off beyond some distance.

Development continued with the addition of probabilistic maps and motion dead reckoning. First the two-dimensional array of the map was changed to a floating-point probability representation.

Individual range sensor readings are processed into a sector of probability data. The point of origin is high-probability unoccupied space and turns into high-probability occupied space at the distance indicated by the sensor. There is also some uncertainty of the angle of the measurement. This is handled by weakening the strength of the probability to the sides of the sector. Sectors are extent-limited to the range of accuracy of the sensors.

The map starts out as a uniform 50% occupancy probability. Each range sector is multiplied over an area of the map corresponding to the position of the robot. As the information accumulates, the area that can be navigated becomes visible as a grayscale. Further techniques could be used to extract hard edges from the map, although I did not end up using any.

I used the differential steering model to implement dead reckoning from the wheel encoder count information. The MCU counts encoder lines even when motors are not being driven. So it was possible to push the robot, and the controller would track its movement. A small moment of accomplishment came when I could push the robot in a circle on the floor, and when it was returned to the same spot, the robot position cursor on the map had also returned to the starting point.

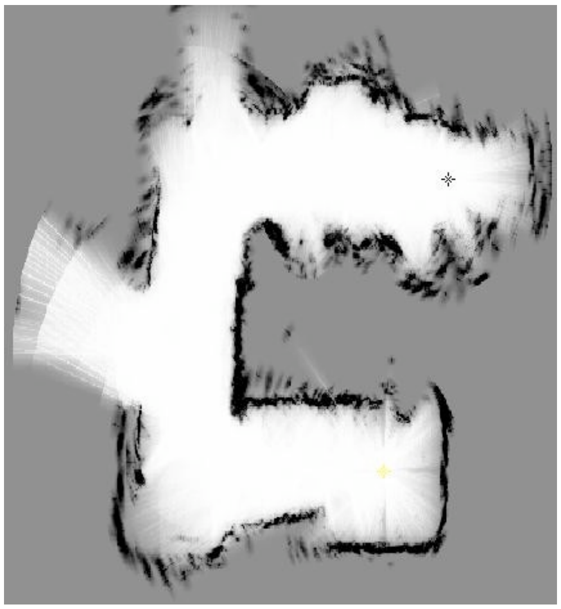
Figure 2

Figure 2 is a map made at this stage of development. There is an open door into a large room at the middle-left of the image. The lower part of the image is in a small bathroom with a door that is not completely open. The upper part of the image is in a walk-in closet with linens, shoes and clothes at robot-level. These give uneven response or no response (like the apparent passageway at the top - that doesn't exist). This map was made with the robot being pushed.

Development continued with a motion planner having three major components (rules). Each rule only references the map that has been created in memory.

1. Forward motion is enabled if the space in front of the robot is known to be clear of obstacles.

2. Looking to either side, the robot will follow a wall (within a limited capture distance) or stay mid-way between walls (as in a narrow hallway).
3. If the way forward is blocked, the robot will look for the closest, largest area of grey (unmapped area) within "eyeshot", and turn in place to face it.

Certainly the control model in this robot is quite simple. Rule 1 has strong reactive control elements. It was the first one integrated, to debug data flow. Rule 3 has strong deliberative control elements. Together they satisfy the control element goals for the robot.

Rule 1 is implemented by observing several rays from the position of the robot out to a distance of 20 cm. They are spread to observe the front of the robot. If any ray encounters space that does not have high probability of being unoccupied, the rule fires, stopping forward motion.

Rule 2 is implemented by observing a ray on each side of the robot at approximately 10 and 2 o'clock to a distance of 60 cm. The rule makes adjustments to steering based on the measured distance on the two sides of the robot. The distance would be terminated with either unknown space or occupied space. Rays encountering only open space are assumed to terminate at 60 cm. The adjustment is a ratio of the two found distances, so the robot stays between narrow walls or follows a single wall at a distance of about 60 cm.

Rule 3 is implemented by observing a spread of rays surrounding the robot to a distance of 400 cm. Obstacles terminate the ray before that distance. The distance to unknown area is found as well as the quantity of unknown on that ray. A quality factor of the quantity divided by the distance is calculated for each ray. The robot is turned toward the ray with the highest quality factor.

In general, the combination is quite successful. The robot can navigate hallways with corners. It can find its way out of simple dead-end traps. There is quite a bit of randomness introduced into the movement planning since there is noise in the mapping data. The robot only bumps into things that are unseen (below view or thin columns like chair legs).

The part that gave me the most difficulty was the third rule. Early on I had worried that the sloppy velocity control of the MCU motor controller would be a problem and it seems that is the case. Instead of being able to accurately turn in place, the robot turns in increments until the rule is satisfied.

Results

The robot was tested in a several rooms. The outline of the primary test area is shown in figure 3. The space is in a home kitchen with some barriers to make it completely enclosed. The doubled line is the face of a dishwasher.

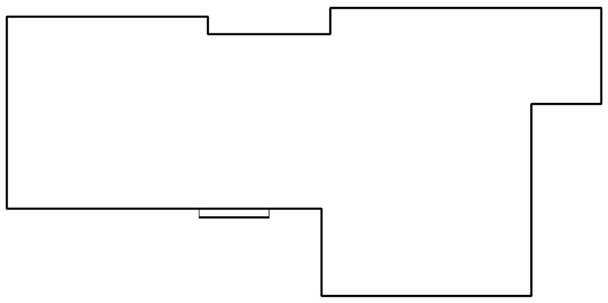
Figure 3

Figure 4 shows four mapping runs. The first three are in the room just outlined. The last one is a hallway that was artificially extended by barriers. The robot was placed in a random orientation each time, so the map orientation changed.

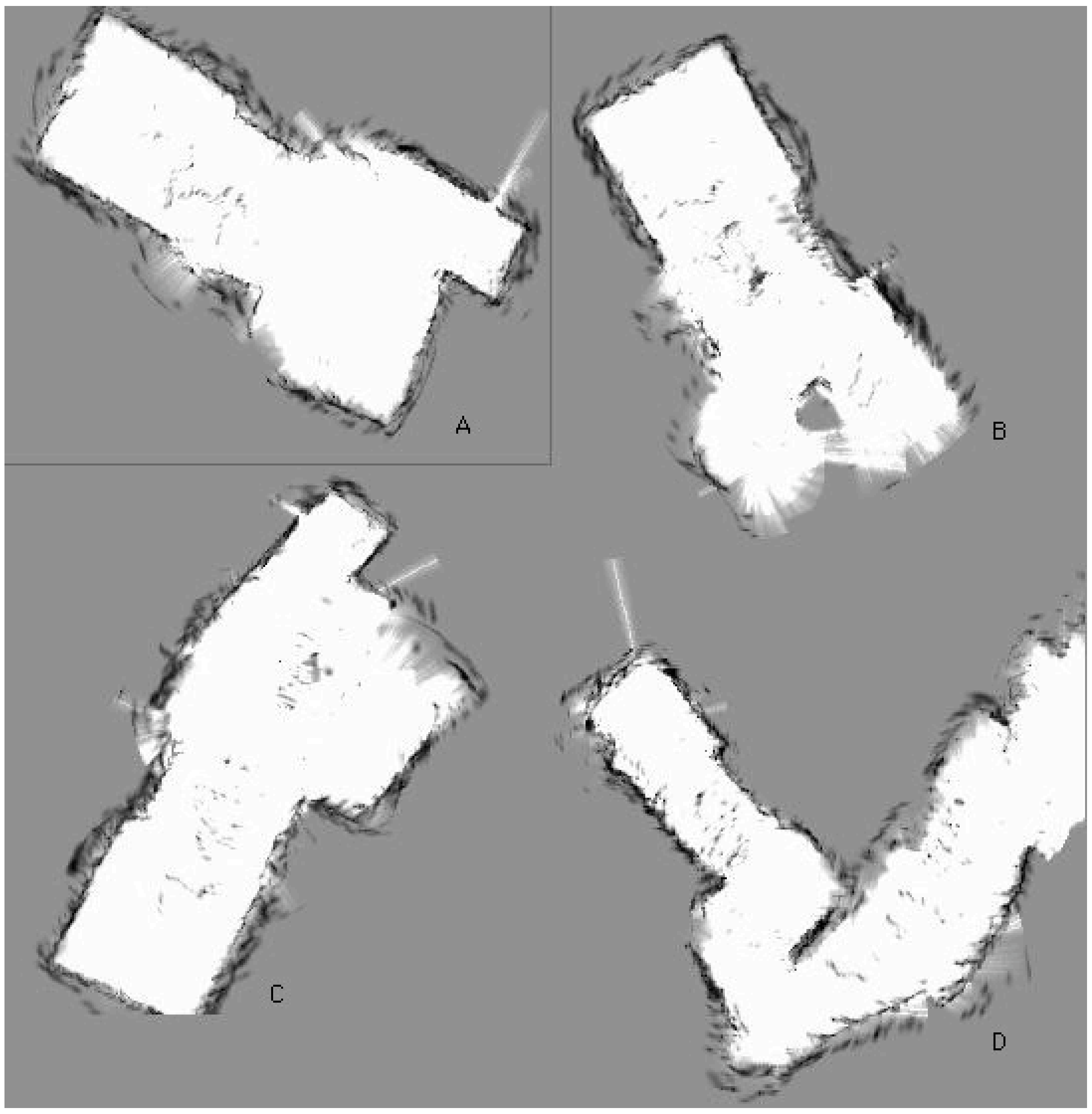

Figure 4

In 4A, the floor was unobstructed. Although this was the most accurate map run, there seem to be a few defects in the map. There are some markings in the middle, but I do not know the mechanism for them at this time. (They may be cells that randomly did not receive mapping probability information.) There is a white spike sticking out that is an unfixed startup artifact from the sensors module. The surface of the dishwasher was not mapped very well. It is shiny white and therefore somewhat specular. At acute angles it likely did not reflect well to the sensors. There is also a slight distortion to the map from compression of the horizontal scale when this image was composited – although that is my error and not the error of the robot. It can be seen in the angles of the room at the top-left.

Case 4B has an ice cooler in the middle (lengthwise in front of the dishwasher) and a round object (a wet-dry vac) in the larger area. (I stopped the robot before it completed mapping that end of the room.)

Case 4C has a round bucket in the larger area. Some of the center obstacles were not indicated as clearly as the walls. This may be because their shapes did not converge as well as the simple shapes of the walls, as the robot moved around them. The problem may also be that they are not as reflective in the IR region as the light-colored walls of the room. It can also be seen that there were problems with the dead reckoning in this case that made the room appear angled.

Case 4D was in a small room and hallway that were artificially extended by barriers to become a winding passage. This case demonstrated traversal of arbitrary passageways. The robot navigated corners and stayed in the middle in places that were narrow.

These images show the robot can map rooms of arbitrary dimensions with reasonable accuracy.

Conclusions

The goal of this project was to learn about autonomous robotics through development of a robot with mapping abilities. The robot operates in a real, but simplified, office-type environment. It includes both reactive and deliberative control elements. It uses distributed processing across MCUs and a main controller.

The robot successfully creates grayscale probability maps of its environment.

Learning about robotics occurred through both the research and the project development.

Bibliography

Arkin, R. (1998) Behavior-Based Robotics. Massachusetts: MIT Press.

Brooks, R. A. (1986) A Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, Vol. 2, No. 1, pp. 14–23.; also (1985) MIT AI Memo 864. Retrieved July 14, 2004 from the MIT Artificial Intelligence Laboratory, personal Web site: http://www.ai.mit.edu/people/brooks/papers/AIM-864.pdf

Everett, H. R. (1995) Sensors for Mobile Robots. Massachusetts: A K Peters.

Jones, J. & Flynn A. (1993) Mobile Robots Inspiration to Implementation. Massachusetts: A K Peters.

Lucas, G. (2000) An Elementary Model for the Differential Steering System of Robot Actuators. Retrieved July 14, 2004 from the Open Source Rossum Project Web site: http://rossum.sourceforge.net/papers/DiffSteer/DiffSteer.html

Thrun, S. (2002) Robotic mapping: A survey. In G. Lakemeyer and B. Nebel (Eds), Exploring Artificial Intelligence in the New Millennium. Morgan Kaufmann. Retrieved July 14, 2004 from Stanford University, Robotics Web site: http://robots.stanford.edu/papers/thrun.mapping-tr.pdf

Varveropoulos, V. (2000) Implementation of an Algorithm for Robot Localization and Exploration. Retrieved July 14, 2004 from the Open Source Rossum Project Web site: http://rossum.sourceforge.net/papers/Localization/PosPosterv4.pdf

Internet publication resources

The Carnegie Mellon Robotics Institute (very comprehensive areas of interest) http://www.ri.cmu.edu/general/publications.html

University of Florida Machine Intelligence Laboratory (wide areas of interest) http://www.mil.ufl.edu/publications/

MIT Artificial Intelligence Laboratory (focus on machine intelligence and robotic interaction with humans) http://www.ai.mit.edu/research/publications/pubs_browse.shtml

University of Washington Robotics and State Estimation Lab (focus on mapping) http://www.cs.washington.edu/ai/Mobile_Robotics/publications-by-type.html

Professor Rodney Brooks (MIT) http://www.ai.mit.edu/people/brooks/publications.shtml

Professor Sebastian Thrun (Stanford University) http://robots.stanford.edu/papers.html

Space and Naval Warfare Systems Center, Robotics http://www.spawar.navy.mil/robots/

The Rossum Project, Open Source Robotics http://rossum.sourceforge.net/papers/

Other resources

Acroname robotics components supplier http://www.acroname.com/index.html

Motorola semiconductor (now Freescale semiconductor) 68HC908KX8 MCU maker http://e-

www.motorola.com/webapp/sps/site/prod_summary.jsp?code=68HC908KX8&nodeId=0
3t3ZGpnLn84498634