

FACILITATING ACTIVE LEARNING

WITH INEXPENSIVE MOBILE ROBOTS

Stephen Paul Linder¹

Brian Edward Nestrick²

Symen Mulders

Catherine Leah Lavelle³

State University of New York at Plattsburgh

Plattsburgh, NY 12901 USA

ABSTRACT

Our educational system must nurture a student's ability to acquire knowledge. Research has proven that active learning, learning promoted by interacting with one's environment, as opposed to lectures, is most effective in developing a student's ability to acquire knowledge. In the computer science domain, active learning can be facilitated by using mobile robots in a collaborative setting. We believe that a context-based, collaborative approach, combined with the excitement, motivation, and real-world experiences provided by a robot, provide a nearly optimal method of teaching students how to acquire knowledge about computer science. However, this methodology has not seen wide acceptance. In order to facilitate the use of robots we have developed an inexpensive wheeled robot that uses common parts and requires only a simple set of tools for assembly. An abstracted Java interface allows students to interact with the robot from any desktop computer using the simple commands provided. Additionally, students can develop sophisticated distributed software solutions that require only simple changes to the interface, and development of software for both the embedded microcontroller and desktop machine. Our approach makes a wide variety of robot-based projects accessible to all level of courses, and for even the smallest computer science departments.

INTRODUCTION

Active learning is the ability to learn through interaction with one's environment. Collaboration is a rich potential component of active learning, since collaboration involves interaction with other people as well as the physical environment. Robotic projects are an effective mechanism for facilitating collaboration and active learning in computer science students. Not only do the robots facilitate **constructivist learning**, they also provide experience with real machines, not the simulated machines so often used in teaching computer science. These real machines give students an opportunity to work with a complex environment where the

¹ *spl@alum.mit.edu*

² *BrianNestrick@netscape.net*

³ *archimedes@alum.mit.edu*

model of the machine is incomplete (Pfeifer 1997). This complexity allows the emergence of behaviors that were not foreseen, making the robot domain even more fascinating for students (Pfeifer 1997; Arkin 1999). When assigning a project, responsibility for integrating the available knowledge into a meaningful solution should be left to the student (Ohlsson 1995), since the emergent behaviors of robots often allow for multiple correct solutions. Students become aware that it is not the writing of the software that is the difficult part, but rather discerning which of many possible solutions is best at solving the problem (Dannelly 1999).

Even with the great benefits to students of including robots in the computer science curriculum (Nourbakhsh 2000), robots have not seen a great penetration into the curriculum. This is perhaps because robots have been perceived as expensive, and if not expensive, mechanically unreliable and providing only a limited possibility for software projects. In an effort to overcome these drawbacks, we have developed an inexpensive robotic car, the **Handy Car** (HC), built from robust commodity parts, with a software package that allows the development of a wide variety of software projects. These projects are appropriate for a variety of classes, from introductory courses to capstone design courses.

Supplementing the mechanical design is an Application Program Interface (API) that supports the use of the RS-232 serial port for real-time interaction between the robot and either another robot or a Personal Computer (PC). A Java-based server allows students to issue commands to the robots, and read sensor values from the robot through an abstracted interface. This facilitates the inclusion of Handy Car in low level software courses because students do not have to learn an additional software development environment to manipulate the robot. In addition, the Server has an Internet interface that allows clients on other PC to logon and control the robot, facilitating the sharing of a robot between multiple students. This distributed architecture also allows students to develop complex projects for capstone courses, using intelligence distributed between the robot, PC and multiple users.

This paper continues with some pedagogic background, a robot description, and a description of several student projects that are appropriate for multiple computer science courses. Detailed information on the construction of the robot, and source code can be found at the following web page: <http://www.plattsburgh.edu/csc/HC>.

PEDAGOGIC BACKGROUND

Constructivist learning assumes that students acquire knowledge by constructing individual models of knowledge (Jonassen 1998). Constructivist learning involves a *manipulation space* which reflects the effects of the student's actions. A student acts, the manipulation space reflects the effects of their actions, the student tries something different, and the manipulation space shows the changes resulting from these new actions. Constructivist learning begins with the question, "What if....?" However, it is essential that students not only pose the questions, but actually test their hypotheses. When students participate in this way, they will retain more information, and develop learning behaviors that are critical for continued intellectual growth (Caine and G. 1994). Students in lectures are not usually the ones posing the questions, and thus do not guide the direction the lecture takes. The rigid sequencing of information imposed by the lecturer may not be the optimal sequence for every student.

Faculty can facilitate active learning by providing students with a rich environment that provides a context for the content of the curriculum. This environment should include an experiential mechanism for exploring the curriculum in the context of the world outside of the classroom. However, the lecture-based courses

commonly taken by students rely on the long held bias in higher education for the “*the sage on the stage*”. This tradition of teaching and learning biases education to the thinking mode, and implies that **content** is all that is important and **context** is unimportant.

Excessive context can, at times, be a hindrance to learning. In physics, novice students are often told to ignore friction, since it greatly complicates the situation. A perceived lack of real-world resources, such as time or money can greatly inhibit a student’s brainstorming of potential solutions to a problem. However, there is a time to ignore context and there is a time to revel in it. Computer programs are written to be run; they are made for a purpose. In the real world, a problem is presented, and then a program is written to solve that problem. Often, teachers attempt to teach in the reverse order: first facts and other content is presented, and then example problems are presented which make use of that recently acquired content.

Involving students with industrial problems that provide context before the content is even presented is one way to promote active learning. If students are aware of the context of a problem, then they can immediately see the applicability and relevance of the content when it is presented. Intertwining theory and practice helps students understand the way they will learn in an industrial setting and provides motivation to excel at their studies(Solomonides and Button 1994). When working in industry they will often have to respond to new challenging practical problems by learning new theory, and then applying the theory to solve the problem. Often faculty have the mistaken impression that they must lead with theory, so that students can then later apply theory to the practice of computer science. Research, however, has shown that leading with practice often provides a greater motivation for students to learn the applicable theory. This **practice driven** approach allows students to develop their own theoretical models autonomously, and does not rely on models that are taught in classrooms (Ohlsson 1995).

Higher education has had an over reliance on traditional didactic methods of teaching. A review completed by Magnesen (Magnesen 1983) gives the following average percentages of material retained in long term memory based on the modality of interaction:

Modality of Interaction	Percent Retained Long Term
Reading	10%
Hearing	20%
Seeing	30%
Seeing and hearing	50%
Discussing	70%
Doing and discussing	90%
Teaching and tutoring	95%

Retention rate of material is particularly important when studying software engineering. The amount of material that the software engineer must become facile with has grown tremendously since the days thirty years ago when computer programs were encoded on punch cards and carried around in boxes. Whereas the syntactical complexity of computer languages has not changed considerably, the ancillary libraries that must be used by the engineer has grown prodigiously. The increase in productivity of software developers has come about in large part by utilizing the large libraries that are now standard with all new programming languages. The retention of all of this additional information more than ever requires students to take an active

interest in their own education. Additionally, students that learn mechanisms for retaining large amounts of technical information early in their career, will be better prepared to create software in today's complex environments.

The higher retention rate of material obtained by *doing and discussing* are recurrent themes in active learning not only because of their positive impact on information retention. *Doing and discussing* also promotes collaboration. Collaboration among students and faculty has been shown as one way to facilitate active learning (Courtney, Courtney et al. 1994; Pappas, Krothe et al. 1998). Collaboration forces students to organize their abstract ideas into concrete sentences in order to convey those ideas to other students. This reworking of one's own ideas creates a more thorough understanding.

The use of robots facilitates this process because the robot in part becomes a participant. The robot grows through the combined efforts of the students. Complexity results in emergent robotic behaviors that excite and invigorate the students (Pfeifer 1997). Students now feel empowered because they can elicit complex behaviors from another physical entity. This perhaps explains the craze for certain toys that elicit seemingly complex response from simple inputs (e.g. the Pokéman Pikachu and the Furby virtual pets). Motivation becomes internalized because the emergent behaviors of the robot give the robot a personality. Students have created an offspring, and they suffer when the robot suffers. Made implicit is that the *intelligence* of the robot reflects upon the intelligence and resourcefulness of the creator.

ROBOTIC DESCRIPTION

The focus of our Handy Car was to provide a low cost, mechanically stable platform for the development of embedded software. The Handy Car and the box for the R/C car kit used as the base for the Handy Car are shown in Figure 1. This approach contrasts to the approach in many introductory robotic classes based on Lego® robots, where the focus is on the holistic design of both robotic hardware and computer software.

Our approach is similar to the three-wheeled Palm Pilot Robot developed by the Robotics Institute of Carnegie Mellon. Instead of desktop computer, a Palm Pilot controls the robot through a serial interface to a Pontech SV203 controller. The controller is used to drive three servo motors and read output from three distance sensors and three shaft rotation encoders. However, we feel that our design is better because students have more cognitive models of our car-based design, and our software development environment is more flexible. The Handy Car software can be developed to run either for the PC or the robot. See <http://www.cs.cmu.edu/~pprk> for more information.

Another example of using robots to aid in the teaching of computer science occurs every year at The Massachusetts Institute of Technology. Each year during the one-month long winter session, students may enter a lottery for one of the coveted slots in "6.270" (pronounced six-two-seventy). Students are given a collection of Lego bricks, electronic sensors, motors, and gears, and the components of a Handy Board. In three weeks, the students, in teams of 2-3, must assemble the Handy Board, design and construct a Lego robot, program the robot, and debug it. The robots then compete against each other on a playing field (table). Students can earn points by completing certain pre-determined tasks, such as picking up tennis balls.

The 6.270 competition differs from our model in several ways. Although Lego robots have become somewhat popular in robotic courses, we have decided to use R/C car bodies instead of having students build their own robot. This allows students to focus all their attention on programming the robot. Additionally, while Lego robots



Figure 1. A Handy Robot with a Plexiglas electronic mounted on a Tamiya car base.

are mechanically fragile, the R/C car base of the Handy Car is mechanically reliable enough to be shared by multiple student teams. If funds do not allow a separate robot for each group, more than one group can share a robot. Sharing of a Handy Car is facilitated by our software. One PC is connected to the Handy Car and acts as a server for the robot. Students then access the Handy Car using clients running on separate PCs. This approach allows multiple students to share one robot in an introductory computer science course.

Another difference between our model and the 6.270 competition is that our students' robots do not compete against each other. Although some students thrive on competition, other students can be intimidated by it or simply don't enjoy it, especially female students. Our approach is more collaborative, where students work with, instead of against, each other.

The next two sections describe the mechanical design of the robot and the software API .

Mechanical Design

The mechanical base of the Handy Car is an extremely stable mechanical environment because it is constructed from a Radio/Controlled (R/C) car kit⁴. These R/C cars were designed to withstand the abuse of children. The car uses a servo motor to steer the front wheels and an electronically controlled DC motor for propulsion. Pulse width modulation is used to control both motors.

As shown in Figure 2, the electronic deck is mounted on the mechanical base using standoffs. All electronics, except for the electronic speed control, are mounted on this Plexiglas deck. The electronics include a 68HC12-based Handy Board, an associated battery pack, a proto board for constructing simple electronic circuits, and IR distance sensors. Three connections are made down to the mechanical base: two pulse width control signals for controlling steering and propulsion, and a digital input

⁴ R/C cars come either as unassembled kits or as prebuilt kits which include a radio controller. We used an unassembled kit because it provides more flexibility and is considerably cheaper.

signal from a Hall Effect switch that counts wheel revolutions. The sensors, breadboard and batteries are mounted using Velcro while the Handy Board is mounted using short standoffs.

This simple design allows a Handy Car to be assembled by any student or faculty in less than two days. Only limited facility with tools is required. A soldering iron is required for electrically connecting sensors, battery pack, and motors to the Handy Board; a hacksaw to cut the Plexiglas for the electronic deck; an electric drill to make mounting holes in the Plexiglas and the bumpers of the car, and a hot glue gun to mount the Hall Effect sensor and magnets. Finally, Velcro strips are used to attach the battery pack and sensors to the electronics deck

A complete Handy Car costs approximately \$470 with the following cost distribution:

Parts	Price
Mechanical Base – a bare R/C car, servo motor, speed controller	\$160
Electronic Deck - Plexiglas, screws and standoffs	\$15
Electronic Controller – an unassembled MIT Handy Board	\$220
Sensors – IR distance sensors and a Hall Effect-based rotation sensor.	\$75
Total	\$ 470

The price will be reduced by \$100 when we replace the Handy Board with a more reasonable priced microcontroller board. Add \$80 for an assembled Handy Board.

Software API

Our software API supports a real-time software interface between the robot and a PC using the serial port of the Handy Board and the PC. Communications is done using a message-based, handshaking protocol based on the SEMI Equipment Communication Standard (SECS) standard. Currently, the PC side of the serial interface is written in Java using the Java Communications (COMM) package: javax.comm. The Handy Board interface is written in C. The actual serial connection can be done using a tethered, wireless IRDA, or radio.

Using our API, control messages are sent to the robot and either solicited or unsolicited data messages are received from the robot. When running the serial interface at 9600 baud, the interface can support 6 data messages per second. Each message contains, a status byte, five distances sensor values, steering angle, and robot speed.

Our Java interface provides a class wrapper around the serial interface, so the complexity of the interface is abstracted. This abstraction allows students in introductory computer science classes to easily interface to the robot. Students can also inherit from this interface and define new commands and data messages. However, the interface code on the Handy Board must be modified to support the new messages.

Supplementing our Handy Car to PC interface is an additional interface that supports communication between the host PC and other client PCs using the Internet. This interface allows several students to share a single robot through the single host computer, facilitating the of the testing of students' software projects.

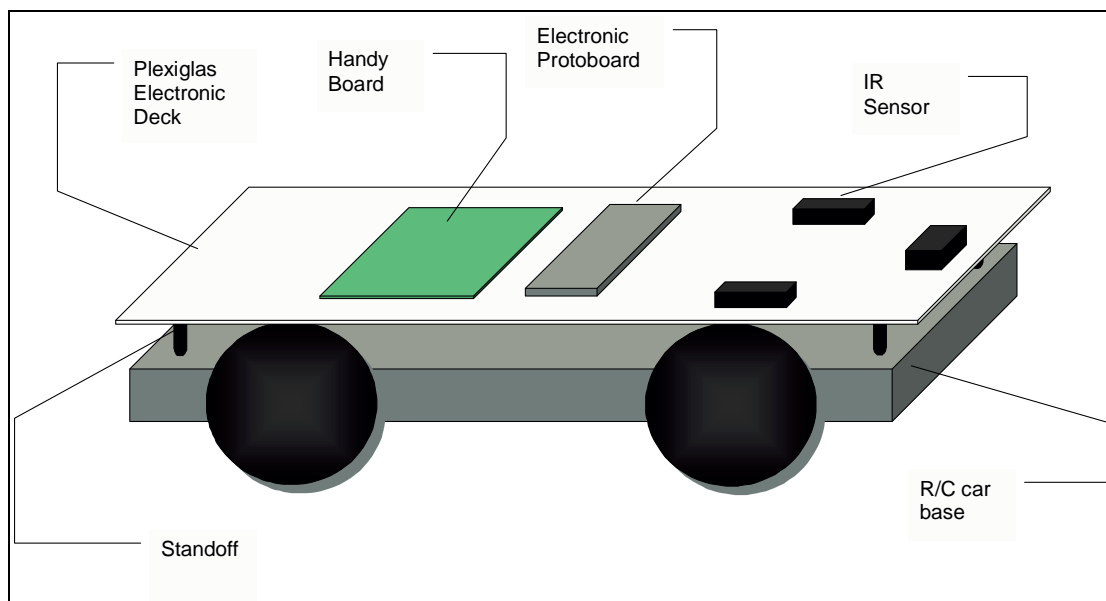


Figure 2. Schematic of robot with a Plexiglas electronic deck mounted on a Radio/Controlled (R/C) car base. The electronic deck contains a Handy Board computer board, IR distance sensors and a breadboard for prototyping simple electronic circuits.

STUDENT PROJECTS

Initially the Handy Car was developed to study the backing up of tractor trailers (Kong 1992; Tanaka 1994). Then, our capstone Software Project course used the robot to develop a maze mapping program where computation was distributed between the robot and supervising PC. This project required students to develop low level software for the robot and high level software for the PC. After the students' success with the Handy Robot it was realized that the Handy Robot was also appropriate for introductory classes.

Next, we describe representative projects for both the introductory level and advanced level courses.

Introductory Courses

In introductory classes the Handy Car can be used by students much like the old Logo Turtle (Papert 1980). The Logo Turtle was programmed by students in the Logo language to draw interesting patterns. Recursion was taught by having students draw complex recursive patterns, including flowers and spiraling n-tuples. Just as for the Logo Turtle, the Handy Car is accessible to introductory courses because the complexity of the Handy Car interface is reduced, through abstraction (Ohlsson 1995; Dannelly 1999). Additionally, by using a robotic car, and not another type of robot, any knowledge models students have developed from operating a real automobile can be used in developing knowledge models for the robots.

The following are suggested projects that use the Java API to control the robot.

Logical and conditional expressions

Logical and conditional expressions can be constructed using combinations of sensor and actuator values. The well known Braitenberg robots (Arkin 1999) use simple logical expressions to combine inhibitory and excitatory sensors to drive the robot. Resulting robotic behavior can be anthropomorphized as aggression, cowardice, and even love. Enabling students to create these behaviors is extremely important in

introductory course where not all students are majors, and not all majors are convinced that computer science will be rewarding.

Definite and Indefinite Iterative Loops

Iterative loops can be used to control the direction and speed of the Handy Robot. Specific sequences of motor commands can be generated by reading them sequentially from a table, or by using the loop index to algebraically generate the motor commands. Students can explore the sequence of motor that minimizes the jerk, the change in acceleration, while giving the fastest response. Any light object (e.g. an empty soda can) placed on the robot can be used a simple visual indication of jerk. Students can attempt to minimize the time taken to traverse a course while attempting to keep the empty soda can on the robot.

Finite State Machines

Finite state machines can be used to construct many complicated robot behaviors. As an example, a state machines can be used to coordinate a robot as it negotiates a K-turn. After creating a narrow dead end street with cardboard, a robot would be required to head down the road, sense the dead end, make a *k-turn*⁵ and return out of the dead end. A simple state machine can be used to coordinate the *k-turn*.

Data Structures

All basic data structures can be incorporated into Handy Car-based projects. As an example, when a programming project requires an unknown amount of data to be saved, a dynamic data structure must be used to save data. Students can use a stack to save steering commands, and then backup to the starting location using the steering commands stored on the stack. Conversely, students can use a queue to replay a pattern of motions. As an example, a robots can be programmed to follow a moving sonar beacon or light beacon, and then the steering commands can be replayed when the beacon is not present.

Advanced Courses

The hierarchical design of the Handy Car lends itself to development of complex projects. Since computation is split between the slower processor of the robot and the more powerful desktop, many computationally intensive projects can be done without purchasing an expensive commercial robot. Additionally, these types of hierarchical systems are prevalent in industry. Common examples include automobiles, a desktop PC⁶, and the space shuttle.

During the Fall 2000 semester, our capstone Software Project team developed a maze mapping product that used a robot to map a maze. A PC-based server issues high level commands (e.g. take next left, take next right) to the robot and collected sensor data. Using the sensor data, the software on the PC was able to generate a map of the maze based on a graph of measured hall lengths and intersections.

The following are suggestions for additional projects.

⁵ A k-turn is a in three-part turn done when there is insufficient room to complete a u-turn.

⁶ A desktop PC contains multiple processors, including processors that control the hard disk drive, the keyboard, and the network interface.

Beacon Search and Homing

In the Spring 2000 semester, students developed a robot that traversed a maze as it homed in on a blinking IR beacon. Students find this problem very challenging because they must write software that arbitrates between obstacle avoidance, wall following and foraging behaviors. An excellent source of suggestions for these behaviors is Arkins(Arkin 1999).

Collecting Balls

Collecting ping pong balls is a popular competition goal. This example of a foraging behavior is difficult for inexpensive robots, but if small beach balls or soda cans are used, the problem is simplified since these objects are easier to detect and manipulate. Balls are pushed into a goal marked by a beacon and scores are given based on the number of balls retrieved.

Parallel Parking and Backing up to a Loading Dock

Robots have been built by researchers to study parallel parking and backing up to a loading dock (Tanaka 1995) . Both problems are familiar to automobile drivers; however, they are not easy behaviors to automate. Papers have been written on using machine learning concepts such as neural networks and fuzzy logic to automate the generation of these complex behaviors (Kong 1992; Tanaka 1994).

Racing in an Oval or Square Track

Racing in a oval or a square track requires the wall following and obstacle avoidance behaviors used in previous example problems. With the added challenge of racing, students must pay particular attention to tuning these behaviors to maximize speed. Races can be held using single or multiple vehicles.

Language and Compiler Design

As a final problem, we propose the implementation of a Logo or Forth like language interface to the robot. These interpretive environments provide an excellent environment for controlling an embedded real-time system.

CONCLUSION

The facilitation of active learning is important for the effective education of all computer science students. Students who consistently develop projects that interact with their environment learn to obtain and retain information better, and learn to organize their abstract ideas into useful models of information. Additionally, interactive projects promote the learning of important collaborative skills. An ideal vehicle for achieving all these goals is the inclusion of robots across the computer science curriculum.

6.270 introduced the possibility of using robots even in lower level engineering courses. Students required a holistic approach to designing their robots. However this holistic approach has limited the adoption of robots to specific robot design classes and artificial intelligence courses. In contrast, we advocated here the use of a low cost, mechanically robust robot, that allows the inclusion of robots in a wider breadth of the curriculum. Our Handy Car, built on a sturdy R/C car base, has allowed our department to utilize robots in both programming and upper division courses. Using the Handy Car does not preclude the retention of 6.270 style courses, but does allow

many more students to experience the positive effects of an active learning environment.

Our experience shows that these robots indeed invigorate students and facilitate active learning. Learning with practice has provided a great motivation for students to learn the applicable theory, and students are now more comfortable in learning autonomously. No longer are students intimidated by a stack of manuals. Finally, our upper division students now demonstrate their projects to captivated lower division students and high school students, providing inspiration and motivation to both groups.

Supplementing this article is our website at <http://www.plattsburgh.edu/csc/HC> where you will find detailed instructions on the assembly and programming of the Handy Car. The website provides a detailed parts list, vendor information, Java source code for the robot server, and example Java client, and the C-code for the Handy Car.

ACKNOWLEDGEMENTS

We would like to acknowledge the supports of the students in our capstone Real-time Software and Group Programming Project courses in developing much of the software required to make the Handy Car run. Brian Nestrick and Symen Mulders, for their undergraduate research project, designed, constructed and programmed the robots, and developed the informational website. If you would like help designing similar systems you can find all their resumes online.

BIBLIOGRAPHY

- Arkin, R. C. (1999). *Behavior-Based Robotics*. Cambridge, MA, MIT Press.
- Caine, R. and C. G. (1994). *Making Connections: Teaching and the Human Brain*. Reading, MA, Addison-Wesley.
- Courtney, D. P., M. Courtney, et al. (1994). "The Effect of Cooperative Learning as an Instructional Practice at the College Level." *College Student Journal* **28**(4): 471.
- Dannelly, R. S. (1999). "Use of a Mobile Robot in a Data Structures Course." *The Journal of Computing in Small Colleges* **15**(3): 85-90.
- Jonassen, D. (1998). Design Constructivist learning Environments. *Instructional-Design Theories and Models*. C. M. Reigeluth. Hillsdale, N.J., Lawrence Erlbaum Associates.
- Kong, S.-G. K., B. (1992). "Adaptive fuzzy systems for backing up a truck-and-trailer." *IEEE Transactions on Neural Networks* **3**(2): 211 - 223.
- Magnesen, V. (1983). A Review of the Finding from Learning and Memory Retention Studies. *Innovation Abstracts*. K. Watkins. Austin, TX, National Institute for Staff and Organizational Development.
- Nourbakhsh, I. (2000). Robotics and education in the classroom and in the museum: On the study of robots, and robots for study. Workshop for Personal Robotics for Education, IEEE ICRA 2000.
- Ohlsson, L. J., C. (1995). "A practice driven approach to software engineering education." *IEEE Transactions on Education* **38**(3): 291 - 295.
- Papert, S. (1980). *Mindstorms*. New York, Basic Books.
- Pappas, V. C., J. S. Krothe, et al. (1998). "Using Collaborative Work Technology to support Active Learning." *Journal of Research on Computing in Education* **31**(1): 49-61.
- Pfeifer, R. (1997). "Teaching powerful ideas with autonomous mobile robots." *Journal of Computer Science Education* **7**: 161-186.

Solomonides, I. P. and B. L. Button (1994). "Improving the quality of student learning: a systems approach." *Engineering Science and Education Journal* **3**(3): 131 - 136.

Tanaka, K. (1995). *Model-based fuzzy control of a trailer type mobile robot*. IEEE International Conference on Fuzzy Systems, IEEE.

Tanaka, K. S., M. (1994). "A robust stabilization problem of fuzzy control systems and its application to backing up control of a truck-trailer." *IEEE Transactions on Fuzzy Systems* **2**(2): 119 - 134.