CORRIDOR NAVIGATION OF A MOBILE ROBOT USING A CAMERA AND SENSORS—MULTI-AGENT APPROACH

by

YUKI ONO

(Under the Direction of Walter D. Potter)

ABSTRACT

This thesis addresses two issues in robotic application: an issue concerned with the verification of how well the existing heuristic methods compensate for uncertainty caused by sensing the unstructured environment, and an issue focusing on the design and implementation of a control system that is easily expandable and portable to another robotic platform aiming to future research and application. Using a robot equipped with a minimal set of sensors such as a camera and infrared sensors, our multi-agent based control system is built to tackle various problems encountered during corridor navigation. The control system consists of four agents: an agent responsible for handling sensors, an agent which identifies a corridor using machine vision techniques, an agent which avoids collisions applying fuzzy logic to proximity data, and an agent responsible for locomotion. In the experiments, the robot's performance demonstrates the feasibility of a multi-agent approach.

INDEX WORDS: Multi-agent systems, Corridor navigation, Collision avoidance, Fuzzy logic controller, Machine vision, Reusable software, Commercial robots and applications, Blackboard architecture.

CORRIDOR NAVIGATION OF A MOBILE ROBOT USING A CAMERA AND SENSORS—MULTI-AGENT APPROACH

by

YUKI ONO

B.A., University of California, Los Angeles, 2000

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2003

CORRIDOR NAVIGATION OF A MOBILE ROBOT USING A CAMERA AND SENSORS—MULTI-AGENT APPROACH

by

YUKI ONO

Major Professor:	Walter D. Potter
Committee:	Suchendra Bhandarkar Beth Preston

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
December 2003

ACKNOWLEDGEMENTS

Academic work is not achieved by a mere individual; rather, it is the art of collaboration. When I first entered the AI Center at the University of Georgia, I neither imagined that I would actually write a thesis about robotics, nor even dreamt of taking a robotics course. However, it all happened after subsequent years by meeting people and working with people. Now I am so grateful for what I have done during my academic years, and what I have done for my thesis work. Therefore, I would like to express my gratitude to the following people for their support and assistance in pursuing my academic career.

Firstly, I would like to thank Dr. Don Potter for giving me an opportunity to work on the robotic project. He has provided me with everything that I need to complete my thesis work and my academic career including the support for robot hardware and invaluable academic advice. I also thank Dr. Beth Preston and Dr. Suchendra Bhandarkar for being on my thesis committee and for spending their precious time on my thesis work.

Similarly, I cannot help thanking Dr. James Smith and Dr. William Graves in the Animal and Dairy Science department for being kind enough to offer me an assistantship opportunity to work in the field of artificial intelligence for years. I thank my friend and fellow students, particularly, Hajime Uchiyama. Without his help and collaboration, my academic pursuit would not have ended successfully. Lastly, I thank my wife Kaori. She has always been the best support since we were good friends, and I will get round to nesting a new home with her soon.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

1. INTRODUCTION

1.1 BACKGROUND

Advances of recent technologies in robotics have already made enormous contributions in many industrial areas. There are uncountable robotic applications found in our society such as surveillance systems, quality control systems, AGVs (autonomous guided vehicles), and cleaning machines (Trahanias et al. 1997; Wijesoma, Khaw and Teoh 2001). These robots do not normally appear in our everyday life, but they play an important role in industries. However, the trend in robotic application is now shifting toward the life of individuals, and robots are now caught in sight more often than ever performing various tasks in disguise. For example, a quadruped robot plays a pet comforting the owner, and a humanoid entertains people demonstrating an ability to mimic human motions (e.g. bipedal walking, juggling with arms). Aside from entertainment, there is also a rapid growth of needs for an intelligent robot in the social and medical fields. Robots are now expected to become the next generation of rehabilitation assistants for elderly and disabled people, and one of the researched areas in assistive technology is the development of intelligent wheelchairs. By integrating an intelligent machine into a powered wheelchair, a robotic wheelchair has an ability to safely transport the user to a destination. This thesis was originally inspired by the need of robotic assistance, and it hopefully leads to future study to build an intelligent powered wheelchair which ultimately assists navigation of disabled people.

Numerous research has been conducted in the field of assistive robotics, and most

studies on intelligent wheelchairs concentrated with developing autonomous behaviors of the mobility aid. Most behaviors exhibited in related literature are the ones concerned with detecting and avoiding obstacles, mapping a surrounding environment, planning safe routes, and navigating a doorway. A wheelchair with such intelligent behaviors often uses sensors such as cameras, ultrasonic sensors, infrared sensors, and laser scanners to interact with an environment, and sometimes builds an internal world model to solve various navigational problems. These intelligent wheelchairs have been developed mostly for adults with severe physical disabilities. Yet, only several have been actually tested and evaluated by disabled people in a real setting. In fact, many research projects dominated by only artificial intelligence and robotics experts have been reported as unreliable in terms of safety due to insufficient experimental results (Nisbet 2002). Many research projects have shown that developing an intelligent wheelchair takes months and even years of effort. In addition, evaluation in real world settings requires another few years, which may or may not result in the approval of the system that ensures the safety of disabled people.

Many engineering maneuvers are developed to solve navigation problems of powered wheelchairs. While the issues in assistive technology have been researched in great depth, there are only a few intelligent wheelchairs commercially available for end users such as *Smart Wheelchair*[1] (designed only for children) which protects the user from collisions and navigates him/her from room to room following the tracks made with reflective tape on the floor, and *SCAD* from Chailey Heritage[2] (Nisbet 2002). Most intelligent wheelchairs are, on the other hand, still under development or only sold to schools and

[1] The CALL Centre (University of Edinburgh). More information available at: http://callcentre.education.ed.ac.uk/

institutes for research purposes. One of the early developments of an intelligent wheelchair was built by Yanco (1995) who introduced *Wheelesley*, a robotic wheelchair system. This semi-autonomous robot travels safely in an indoor environment using various sensors. Also, with the graphical interface, users can easily navigate the wheelchair by selecting a simple instruction which represents a course of several navigational tasks. *NavChair* is one of the most successful intelligent wheelchairs. It was developed at the University of Michigan (Levine et al. 1999). The tasks of this robotic wheelchair consist of the following three modes: (1) obstacle avoidance, (2) door passage, and (3) wall following. The control system automatically changes the mode according to the environmental surroundings. The *TAO* series developed by Applied AI Systems Inc. are famous intelligent wheelchairs for exploring in an indoor environment (Gomi and Griffith 1998). In addition to the tasks performed by the *NavChair, TAO-1* and *TAO-2* also have two additional tasks: (1) escape from a crowded environment and (2) perform landmark based navigation. Currently, *TAO-7*[3] is in use. At the KISS Institute for Practical Robotics, *TinMan II* (Miller 1998) is in the early stages of assistive robotics development. The *TAO* and *TinMan* series have been sold to other institutions such as the MIT AI Lab (*Wheelesley*) and the University of Rochester (Yanco 1998) as prototypes for intelligent wheelchair development. *Rolland* (the Bremen Autonomous Wheelchair) assists the user in obstacle avoidance and door navigation (Lankenau, Röfer and Krieg-Bruckner 2003). *MAid* (Mobility Aid for Elderly and Disabled People) was experimented with crowded environments (e.g. a railway station) and successfully navigated in heavy passenger traffic (Prassler et al. 2001).

---

[2] The company's website: http://www.southdowns.nhs.uk/directory/chailey/
[3] Information is available at the website: http://www.aai.ca/

In the development of an intelligent wheelchair, we consider roughly two kinds of tasks, safety-oriented tasks and navigation-oriented tasks. The safety-oriented tasks include behaviors such as collision detection, obstacle avoidance, lane (or corridor) detection, wall following, and door navigation, and all of them ensure collision-free navigation for wheelchair users. This group of behaviors generally exhibits behaviors in a reactive manner. The navigation-oriented tasks, on the other hand, involve relatively heavy cognitive tasks compared to the safety-oriented tasks. The behaviors such as environmental mapping and route planning are typical examples of the navigation-oriented tasks. There is no choice over which group of tasks is more important than the other. However, the degree of autonomousness of a wheelchair may affect the prioritization of tasks. For example, developing a semi-autonomous wheelchair usually leaves high-level decisions to the user and thus prioritizes the safety-oriented tasks. The thesis is founded on an aspiration of building a semi-autonomous intelligent wheelchair. Therefore, the safety-oriented tasks are considered as the first priority, particularly collision avoidance and corridor recognition. A small mobile robot is used as a test bed in the unstructured indoor environment for experimenting with the robot control program designed possibly for an intelligent wheelchair.

1.2 PROBLEM STATEMENT

This thesis addresses two issues in robotics. Firstly, one issue is concerned with the verification of how well existing heuristic methods can compensate for the uncertainty caused by sensing the unstructured environment. A typical problem in dealing with uncertainty is often found in mobile robot navigation. In this thesis, an autonomous

mobile robot navigates itself in a hallway. The robot demonstrates solutions to typical navigational problems, *corridor (lane) detection* and *collision avoidance*, in an indoor (office-like) environment. The second issue involves the design schema of robot control software. A proposed framework is to design and build a robot control program that is independent of the system platforms and easy to expand for future study.

In order to achieve successful navigation in a narrow hallway, a robot must exhibit fundamental abilities such as recognizing a corridor and detecting and avoiding collisions. The autonomous robot equipped with agents performing such tasks requires information about the environment where the robot is situated. The robot used in this thesis is realized using a minimal set of sensors such as a camera and infrared sensors. The camera captures the front view of the surrounding environment, and the infrared sensors detect objects in the nearby vicinity. The employed agents are capable of handling uncertainty by compensating for the inaccuracy of the sensor data using a-priori knowledge and heuristic methods such as fuzzy logic. The corridor navigation agent, for example, processes a captured image and identifies a corridor using machine vision techniques. The common strategy is *lane detection*, which includes robot (or vehicle) localization as well as path extraction. Extracting a path from the image using edge information, the agent determines the relative position between a robot and the extracted path (Bertozzi, Broggi and Fascioli 2000). The collision detection agent uses the numerical range data acquired from proximity detectors or ranging sensors to provide information necessary for the robot to avoid collisions. Employing fuzzy logic enables the collision-free navigation task with a minimal hardware system. A mobile robot with these intelligent agents exhibited successful collision-free navigational behaviors in our unstructured

indoor environment.

Recent robotic technology has drastically evolved mostly due to the enormous advancement of personal computers. About thirty years ago, the famous Moore's Law[4] predicted today's revolutionary improvement of silicon chips. Similarly, computer software has also become much more intelligent, and some have demonstrated human-level expertise. This hard-soft synchronization seems to be the key for a rapid growth in the robotic industry. Numerous robots are now available not only for the sake of reducing human cognitive and physical tasks, but also amusing, helping and assisting people. It is a big step in the relationship between humans and robots because we no longer operate but "interact with" robots. We now feel that these robots are very close to our everyday life.

Likewise, robotics in the research domain is also taking a big step between researchers and robots. Robotics researchers used to be dominated by only robotics or artificial intelligence experts who built and programmed a robot from scratch. However, due to the recent developments within the robot industry, building a robot has become much more effortless with the aid of commercially available robot kits. Today's introductory robotics course does not have to depend on the intricate knowledge of technical details. This brief statement does not jump to the conclusion of neglecting these essential skills and knowledge, but it only suggests that the commercial robot kits quickly involve students in the real problems of robotics. This consequently enables learners to focus on their ingenuity rather than being stuck on technical problems.

There are more reasons to make use of commercial robot kits. Using these kits allows

[4] The observation made in 1965 by Gordon Moore, co-founder of Intel that data density doubles approximately every 18 months for the foreseeable future.

us to possibly reuse the robot and the robot control program. Building a robot usually requires the following steps: purchasing or manufacturing individual parts, wiring the electrical circuits and chassis, and assembling them all into one piece. Therefore, once we have built a robot, it is obviously difficult to disassemble, reassemble and reuse it in different robotic applications. Likewise, the control software specifically designed for a particular robot is most likely incompatible with other robots. The use of a commercial robot kit may simplify the design of control software since the kit is usually provided with useful tools. It is also a good start for building a program capable of being reused for other robotic applications.

Implanting a control system on a different robot, it is apparently legitimate to make a statement that the reuse speeds up the entire process of robot production. Developing a computer program is generally a time-consuming task, and developing a robot control program to deal with a machine embedded in the physical world is even more challenging than the common computer programs that only deal with abstract entities. To evaluate the performance of the tasks specified in a program, no matter what the tasks are, the software must be integrated into a robot and tested in the physical environment. Therefore, the robot, the program, and perhaps the environment must be arranged for the complete evaluation.

Reusing a program which has already been tested and proven to perform certain tasks (at least in the specified condition) can save enormous time and cost in building and testing a robot. In the field of robotics and artificial intelligence, there are numerous papers presenting solutions to tackle hard problems, explaining their approach with various methods and techniques. While most studies have demonstrated the feasibility of

algorithms or behaviors, not many papers have extensively discussed the reusability or expandability of control software independent of the system platform or the robotic hardware. This thesis focuses on this issue by proposing a framework for building a mobile robot that will standardize the system compatibility and expandability with its incremental design, aiming to future research and applications.

2. HARDWARE DESIGN

2.1 ROBOT KIT

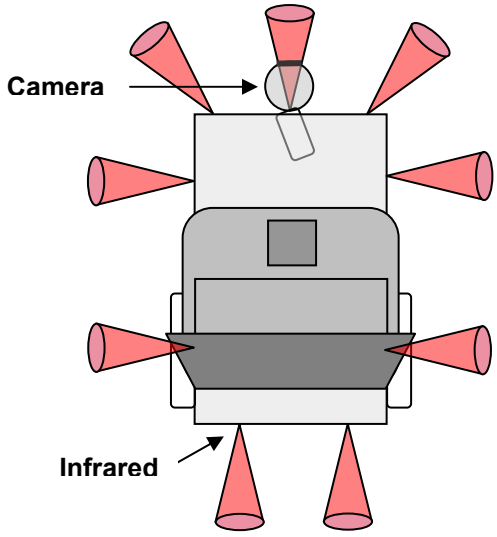
The hardware used in this experiment is a commercial robot kit called the *ER1 Personal Robot System*, supplied by evolution robotics™[5]. The robot kit includes the control software, aluminum beams and plastic connectors to build a chassis, two assembled nonholonomic scooter wheels powered by two stepper motors, one 360 degree rotating caster wheel, a power module, a battery (12V 5.4A), and a web-camera. The experimental robot also carries additional accessories, nine infrared sensors and extra beams and connectors for reinforcement. A laptop computer, Dell™ *Latitude C640* (Intel® Mobile Pentium® 4 processor 2.0GHz with 512 MB RAM), is used as a controller device, and Windows XP Professional is loaded as the operating system.

Figure 1. Customized ER1 Robot

The bundled software that comes with the kit provides various tools for the users to operate the robot with its simple interface such as computer vision, hearing, speech, networking, remote control, email, and some autonomous behaviors. However, the furnished high-level behaviors have no flexibility in customization at the algorithmic level of behaviors which in many cases requires programming for modifications.

[5] More information available at: http://www.evolution.com/

Therefore, the experiments have been conducted without using the bundled software. Unlike the software, the hardware of the ER1 robot kit empowers users to customize the robot for their objectives. The reconfigurable chassis enables us to design a purposive mobile robot, and the extensions (extra cameras, sensors and grippers) can be easily added to the system if necessary. The purpose of this experiment is to build a robot as a test-bed for the future wheelchair project, so the autonomous robot is modeled after the typical powered wheelchair with two independent wheels.



One web-camera is mounted in front of the vehicle, and nine infrared sensors are installed circling 360 degrees around the vehicle.

Figure 2. Sensor Arrangement

2.2 SENSORS

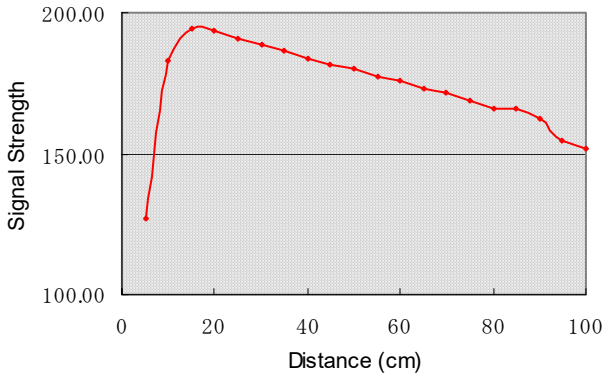
In this experiment, nine infrared (IR) sensors and a single web camera are used and gather information about the environment. Figure 2 depicts the arrangement of sensors installed on the robot. The camera, *Logitech® QuickCam® Pro 4000*, (Figure 4 Left) is mounted in front of the vehicle capturing the front view as in Figure 3. The 160 x 120 32-bit RGB image is updated and saved in memory at the rate of 10 frames per second. The camera is connected to the PC through a USB (Universal Serial Bus) port

Figure 3. Captured Image

and used mainly for recognizing a path in the hallway. The IR sensors enclose the rectangular robot fairly evenly for 360 degrees as in Figure 2. Three sensors are bundled together as one piece. The bundled pack incorporates three modulated infrared sensors, and each and every sensor can be individually manipulated by the PC via one USB port. The *evolution robotics IR sensor pack* (Figure 4 Right) is manufactured and provided by *evolution robotics*, the vendor of the ER1 robot kit. The IR sensors are sold separately as an extra peripheral. The typical distance measurement of the IR sensor is shown in the graph (Figure 5). The sensor is measured against a smooth white wall under fair lighting conditions assuming the hypothetical corridor environment. The possible sensor value ranges between 0 and 255 although the graph only presents the range between 100 and 200. According to the measurement, the distance between 15 cm and 100 cm should reliably be acquired in reasonable ambient lighting conditions. Normally, the larger value the sensors read, the more closely the robot finds the source object. However, the graph shows that within the range below 15 cm, the sensor value drops rapidly and starts to look like a longer-range reading. This

Logitech QuickCam Pro 400 (Left)
evolution robotics IR sensor pack (Right)

Figure 4. Photos of Sensors



Measured sensor signal (Y-axis) maps the distance (X-axis) in cm.

Figure 5. Signal (digital) /Distance Mapping

can be disastrous if a robot is slowing down as it approaches a solid object while the sensor value is reaching below the minimum range, and then at the next moment, the robot suddenly misinterprets the apparently long-range reading driving full-speed into the object. The easiest solution is to crossfire the sensors so that each sensor covers the other's dead zone[6]. The situation can also be avoided by employing sensor fusion (Wijesoma et al. 2001). Behaviors such as collision detection and obstacle avoidance are designed to perform tasks based on the information given by these sensors. Behaviors are extensively discussed in Chapter 4.

2.3 SURVEYS

There are many sensors considered to be useful in mobile robot navigation. For example, one kind of sensor is used to physically interact with and make changes based on the environment (e.g. touch, ultrasonic and infrared sensors), and another kind is used to perceive the environment without interaction (e.g. vision, temperature and compass sensors). There is also a kind of sensor that measures or approximates the internal state of the robot in relation to the environment (e.g. shaft encoder and gyroscope). In this section, we only focus on sensors used for the experiments or concerned with the future study of building an intelligent wheelchair.

A vision system is considered as a *passive sensor* and has fundamental advantages over the sensors that are considered as *active sensors* such as infrared, laser, and sonar sensors (Bertozzi 2000). Passive sensors such as cameras do not alter the environment by emitting lights or waves in acquiring data, and also the obtained image (data) contains more information (i.e. substantial, spatial and temporal information) than active sensors.

[6] *Demystifying the Sharp IR Rangers*: http://www.acroname.com/robotics/parts/R48-IR12.html

However, visual information can be easily fooled by the weather or the environment (i.e. night, back-light, foggy and rainy weather). On the other hand, active sensors are robust in severe environmental conditions and any computation is relatively inexpensive.

Although cameras are widely used in various robot applications, using a single camera is not a major solution, especially in mobile robot navigation. Stereo vision (or stereoscopic vision) using two or more separate cameras (Mazo et al. 2002; Goldberg, Maimone and Matthies 2002; Asensio, Martínez and Montano 1998) is the most widely accepted solution in robot navigation. With stereo vision, we can see "where" objects are in relation to our own bodies with much greater precision, especially when those objects are moving toward or away from us in the depth dimension. Also, stereo vision can be realized with little expense with no entangled installation, at the cost of doubling the energy consumption and allowing comparably expensive computation. Besides the resource problems, *correspondence problems*[7] (matching points between two input images) are the known impediment and virtually impossible to solve without errors (Hirschmüller 2002).

Omni-directional (or panoramic) cameras are also a part of the mainstream in robotic navigation. Technically, omni-directional vision can be achieved in various ways. For example, there are cameras with extreme wide angle lenses (fish-eye), cameras with hyperbolically curved mirrors mounted in front of a standard lens (*catadioptric* imaging), sets of cameras mounted in a ring -like fashion, or an ordinary camera that rotates around an axis and takes a sequence of images that cover a field of view of 360 degrees. With an ability of capturing the wide range of the surrounding environment, the omni-directional

[7] Vision algorithms typically deal with correspondence problems in processing multiple frames over time. Stereo vision needs to solve additional correspondence problems at each frame.

camera is acknowledged as one of the most powerful tools in locating nearby obstacles and their relative positions (Hundelshausen, Behnke and Rojas 2002; Argyros et al. 2002; Matsumoto et al. 1999), tracking moving objects (Stratmann 2002), and localizing a robot in the environment (Paletta, Frintrop and Hertzberg 2001). Despite the comparable advantages, the Omni-directional cameras have drawbacks of cost performance (either in expense or labor) and complexity in developing software in which vision algorithms have to account for the specific properties of the particular omni-directional imaging sensor setup at hand.

While relying on a single camera is not as powerful as the former approaches in functionality, there are a fair number of research projects done using only one camera (mostly in combination with other types of sensors) because the advantages of minimizing cost and having easy installation are attractive and worthwhile. The usage of a camera is varied in projects; one used a camera for recognizing a path including corridors and roads (Broggi and Bertè 1995; McDonald, Franz and Shorten 2001) and detecting dynamic and static obstacles (Trahanias et al. 1997). The other extracted features such as faces, signs, and landmarks using a camera (Röfer 1997; Mazo et al. 2002; Schilling et al. 1998). Most research projects use cameras to obtain auxiliary evidence for high-level decision-making while the essential information regarding safety is mostly dependent on active sensors.

Measuring distance of nearby objects and walls is the most necessary and important task for autonomous mobile agents. Most previously conducted research has used at least one type of active sensor (ultrasonic, infrared or laser) for ranging purposes. More than ninety percent of the studies reviewed for this thesis use ultrasonic (sonar) sensors, which

are thought of as the most widely accepted sensor in mobile robot navigation because of its cost performance. Various transducers are commercially available at reasonable prices (e.g. Polaroid series). In typical configurations, sonar sensors are mounted in a ring around the vehicle (Katevas 1997; Lankenau 1998), or sometimes the array of sensors only covers the front side of the vehicle (Simon 1999).

Making use of the virtue in ranging, ultrasonic sensors are often used for obstacle avoidance where the robot needs to detect static objects possibly blocking the navigation route. Sonar sensors measure distance against target objects in good approximation (covering more than 3 meters), practically regardless of any materials, but also have severe drawbacks inherent to the principle of ultrasonic sensors. Well-known sensor cross talk is due to the wide-angle emission cone of sound waves, which causes directional uncertainty. Also, the transducer sometimes does not receive reflected sound waves when the angle of a tilted object surface is too large (Borenstein and Koren 1988). Some novel research, on the other hand, has been conducted overcoming the shortcomings. Borenstein (1991) invented the VFH (Vector Field Histogram) Obstacle Avoidance System[8] which is employed in the NavChair (Simon et al. 1999). Ushimi et al. (2002), for example, simulate the sonar-based method to navigate an autonomous robot safely in a dynamic environment avoiding coexisting multiple moving obstacles.

Sonar sensors so far appear to be the best solution in ranging because of the cost performance; however, laser-based sensors in fact are superior in range approximation and found in many practical areas. “Laser” stands for Light Amplification by Simulated Emission of Radiation, and the laser scanner is basically measuring reflected light (or

[8] Obstacle avoidance methods based on ultrasonic sensors, accomplishing with the *histogram grid* world model that is updated by rapidly firing 24 sensors around the robot during motion.

emitted photons) of a specific frequency in a straight beam originating from the scanner itself. Laser products are commercially available (e.g. Acuity, SICK and SUNX) and they are commonly used in robotics projects. There are some advantages and drawbacks to using laser-based sensors. First of all, laser-based sensors can extract information more than just distance. For instance, a laser scanner is often used to extract topological information making the best use of its ability to identify the textures of an object's surface and its precise range approximation. Also, in ranging the laser range finder, for example, has considerable advantages over ultrasonic sensors in many aspects such as instantaneous measurement, superior range accuracy, and precise angular resolution. In fact, Wijesoma et al. (2001) presented the advantage in the directionality problem using narrow beam sensors over ultrasonic sensors that have wide emission angles. At the same time, the laser range finder has a fatal disadvantage; the scanner misses transparent objects such as glasses and windows (Jensfelt 2001). In addition, the fancy functionality may not be worth spending in exchange for the overpriced equipment. Examples of using laser-based sensors are found in many robotics papers including the field of assistive technology (Prassler, Scholz and Fiorini 1999; Fod, Howard and Matari´c 2002; Arras, Tomatis and Siegwart 2000).

With respect to cost performance, infrared (IR) sensors are another major solution in mobile robotics. IR sensors have limited usage; they are normally used as proximity detectors rather than range finders because of their limited (short) range and their susceptibility to ambient light interference. IR sensors are also known for their non-linear behavior (see Figure 5 in the previous section) and their reflectance dependency on the surface of a target object (Benet et al. 2002). However, the shortcomings are not as

serious as those of the other active sensors, and a number of research projects have shown the significant improvements on sensor performance by compensating for the uncertainty caused by the sensors.

IR proximity detectors work somewhat similarly to laser range finders. Infrared light (possibly pulsed) is emitted and the detector measures the reflection of the back-scattered light. Although Sharp IR detectors are currently the most inexpensive commercial products winning a reputation, but they are still "dumb" sensors and need some intelligent compensation. In mobile robot navigation, infrared sensors are mostly used in the safety-oriented tasks such as collision detection and obstacle avoidance because of faster response time and lower cost (Benet et al. 2002). Most studies mix infrared sensors with other sensors in order to optimize the tasks (Martinez, Tunstel and Jamshidi 1994; Röfer 1997; Prassler et al. 1999; Mazo et al. 2002) while some achieved one or more of the tasks only using a set of IR sensors (Kube 1996; Maaref and Barret 2002).

3. SYSTEM APPROACH

3.1 INCREMENTAL DESIGN

The ultimate goal of our robotic experiments is to build a controller which can be used in the future study of a robotic wheelchair. In order to build such a robust and compatible program, we must build up the program as a complete system with a set of complete behaviors, which enables the robot to be tested in the real world environment. Rodney A. Brooks at the MIT AI Laboratory suggested in his famous article that building complex robots (he calls *creatures*) which coexist in the world with humans must be incrementally built in the same manner as biological evolution (Brooks 1991). For instance, a single-cell amoeba which wanders the world without any goal and a human who exhibits intelligent behaviors are both complete biological systems although there is a difference in the degree of intelligence. During the astronomical time span, biological evolution on earth started from the lower intelligence of amebas and now has ended up with the human level intelligence up to this point. Brooks' idea of building a robot mimics the process of evolution. The concept of this incremental design helps the entire project of building an intelligent system to advance toward to the goal steadily one step at a time.

3.2 SYSTEM ARCHITECTURE

The system architecture is an abstract design that organizes the system components. In the recent robotic literature, most autonomous robots employ a layered architecture.

There are roughly two types in decomposing the system into layers, functional-based layers and behavior-based layers. Nowadays, the trend in layered architecture is Brooks' *subsumption architecture*, in which the system is decomposed into task-oriented behaviors (Brooks 1986). In the subsumption architecture, the independent behaviors exercise their tasks (from sensing to acting) in parallel. Therefore, the failure of one behavior does not interrupt the entire system execution. The independence of behaviors also gives the capability of easily adding more behaviors to the system in an incremental manner. Each behavior can either *suppress* or *inhibit* the input/output of other behaviors to interact with the environment, which causes the emergence of a high-level intelligent behavior without giving the robot specific instructions of what to do to achieve that particular behavior. Also, the absence of a central reasoning protocol, no symbolic representation of the world model, and the direct control of actuators by a behavior are well-known distinctive characteristics of the subsumption architecture (Brooks 1991). Although each behavior is independent, the ability of influencing another behavior eventually makes the system very complicated, and adding another behavior may thus require enormous efforts. In addition, because of the emergent characteristic of behavior-based systems, the

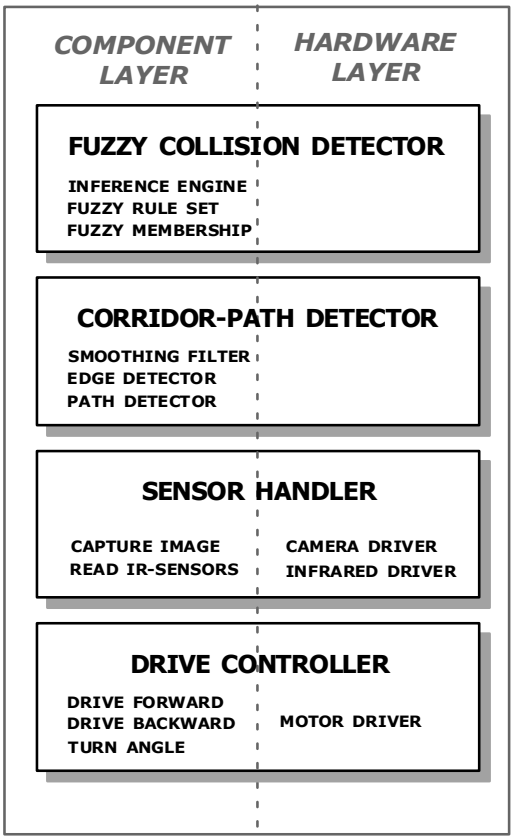


Figure 6. Layered System Architecture

complexity in analyzing the result of emergent behaviors may also cause a problem in modifying and configuring the system. The classical hierarchical approach had been, on the other hand, dominating the robotic trend for decades until the rise of the subsumption architecture. Unlike the behavior-based decomposition of the subsumption architecture, the traditional layered architecture decomposes the system into functional modules such as *sense*, *plan*, and *act*. This type of architecture has the advantage of having easily separable functional modules that are associated with an intuitive paradigm in designing the hierarchical architecture. However, it is often noted that the system is hardly modifiable once the hierarchy is defined since the functionality of modules is limited to contribute to certain behaviors (Liscano et al. 1995).

The robotic system architecture used in this thesis (Figure 6) consists of two layers taking the advantage of the former two types. Basically, the system has four task-oriented agents in the behavior-based like structure. Each agent is composed of two functional layers, *Hardware Layer* and *Component Layer*. The Hardware Layer is a collection of modules communicating with the robot's hardware devices such as a camera, infrared sensors and motors. The Hardware Layer is implemented with Visual C++ .NET since the ER1 kit is provided with the development environment that specifies the language. The SDK (Software Development Kit) already contains libraries to help in accessing the hardware components of the robot, which reduces the amount of redundant effort. This layer functions as a bridge between the upper-level layer and the hardware. The Component Layer contains the intermediate functional modules which constitute the higher-level behaviors as *agents*. One module can be shared by two or more agents, which reduce redundancy in coding. The Component Layer is implemented with Java™

Technology (Sun Microsystems, Inc). Programming in Java has enormous advantages in building a robot control application. The next section explains why.

3.3 PLATFORM INDEPENDENCE

Java is a fairly new programming language and has not been widely used in robotic applications. This is mainly because Java has been incapable of coping with real-time problem solving because of its slow execution. It is true that Java is relatively slower than native languages such as C and C++, while a robot control program must be "time-sensitive". However, discounting the time issue, we find Java has significant advantages over C/C++ in many areas.

- The program written in Java (including GUI) is known to run almost anywhere regardless of operating system or hardware, and the development tools are also available in most platforms. "*Write once, run anywhere*"

- Java has the ability to operate our own or third-party optimized C/C++ libraries (*dll* in Windows, *so* in Linux, etc.) with minor adjustments or simple wrappers.

- Java supports multi-threading at the language level. Real-time applications often require parallel processing on a single CPU.

- Built-in security for safe and reliable network communications via TCP/IP has efficient and robust native support in Java. Compact bytecode enables downloadable Java applications (applets).

- Java has automatic garbage collection for memory leaks and also has simple native thread creation and automatic cleanup for keeping the program from code leaks.

- Direct interfacing with USB devices (the *javax.usb* package) is available (coming

soon for Windows). Java implementation of USB is under development with the support of Fujitsu, IBM, and Sun Microsystems.

- Many APIs (Application Programming Interfaces) for Java are already available for free.

The recent advancement of computer technology enables us to build a Java-based control program which can solve problems in real-time. Although it is still slower than native programs, it does not bother us if the program completes the job in reasonable time. Java provides more distinctive strength in controlling a mobile robot system. Most robot control programs in the robotics literature seemed mostly hardware specific. It is mainly because the uniqueness of hardware and operating systems limits on the programming language and the development environment specific to the platforms. Meanwhile, *platform independence*, one of the most attractive features in Java, allows the control program to operate similar robots on various computer platforms and operating systems with minor modifications.

Having this feature in the robot control program, we can test the program on a smaller scale prototype. Using a test-bed for a robot control program is especially needed if the robot is a large intelligent system equipped with many features. Another advantage of using Java in mobile robot control is the availability of APIs. Sun Microsystems already provides a number of useful tools for free which are unavailable in C/C++. In the meantime, many research institutions, companies, and even individuals distribute miscellaneous APIs with or without charge. In this thesis, those useful APIs are used in the control program. For example, the fuzzy collision detection agent uses the *NRC*

FuzzyJ Toolkit freely distributed by the National Research Council of Canada[9]. This fuzzy toolkit API provides the capability of handling fuzzy concepts and reasoning. Using these APIs usually maintains system compatibility.

[9] More information available at: http://www.nrc-cnrc.gc.ca/

4. SOFTWARE DESIGN

4.1 ROBOT CONTROL ARCHITECTURE

In order to execute multiple tasks on a single processing unit, the robot control architecture must be carefully designed in a way that the robot would choose the right action among many candidates. In Chapter 3, we discussed the classical hierarchical architecture and Brooks' subsumption architecture with respect to the system organization. In this section, we discuss issues within the robot control spectrum rather than the system design. The control method theoretically lies between two extremes, the *planner-based centralized approach* and the *decentralized purely reactive approach* (Mataric´ 1992). The former is a control method which makes a global decision on the robot's action by building a complete internal model of the environment using a-priori knowledge and perceived data. On the other hand, the reactive approach normally maintains no internal model and locally decides the robot action based on the sensor inputs using simple if-then rules. In the recent robotics literature, non-extreme control models such as hybrid[10] and behavior-based[11] systems gained popularity because of their moderation that is relatively applicable to the realistic situations which usually require real-time sensitivity and planning capability.

Various methodologies (e.g. behavior-based, blackboard, and agent-based systems) are found in many projects on mobile robot navigation. In terms of the control

[10] The architecture employing a reactive system for low level control and a hierarchical system for higher level decision making.
[11] Sometimes referred to as subsumption systems. The subsumption architecture (Brooks 1986)

mechanism, the subsumption architecture seems valid and attractive because of its parallelism in a decentralized fashion and also because of its instantaneous decision-making process. However, behavior-based autonomous robots are hardly seen beyond research domains because of the structural complexity (designating the inhibition and suppression among multiple behaviors could be a complex and messy job) and the verification difficulty (due to the decentralized nature the robot may express highly unexpected (emergent) behaviors which makes it difficult to analyze the robot's behavior patterns). Besides, since the truly distributed model requires multi-processing units, the concept does not completely match the objective of using a commercial robot kit as the robot's framework. Therefore, the behavior-based system may not be the perfect model for building the robot control program this time.

Meanwhile, the blackboard architecture (Corkill 1991) also provides some attractive features for mobile robot navigation. The concept of a "blackboard" is a metaphor for information sharing among multiple heterogeneous problem-solving agents. The blackboard system reasons about the robot's up-to-date situations posted on the blackboard and selects appropriate actions by processing symbolic information using production rules (Liscano et al. 1995). One reason for using the blackboard architecture in robot navigation is its adaptability for the application needed to make dynamic control decisions. The activation and execution of agents are dynamic, and thus there is no formal algorithm for controlling the robot behaviors. The blackboard system enables the real-time activation of the most appropriate behavior in response to sensory interpretation. Another reason is that diverse and specialized knowledge representations are possible within a common data structure. This flexible representation of blackboard information

---

addresses a specific architecture within the behavior-based framework.

makes any type of problem-solving agent available to incrementally solve a complex problem. This is, in turn, to state that an agent (or a module) can be an expert system, a neural network, fuzzy logic controller, or a conventional algorithmic procedure. The representation data types on the blackboard can also be of any form such as a vector, a formula, a string and a complex object. In this experiment, this is important because each robot behavior deals with different types of sensory inputs and solves various problems in order to achieve the goal. However, because of the presence of a global database, reactivity to the dynamic environment may not be instantaneous. Also, the existence of a control module (sometimes called an inference engine) may imply that blackboard systems are not as robust and reliable as behavior-based systems. Once the control module stops functioning, the whole system collapses. On the other hand, having a malfunctioned behavior (or agent), the subsumption system still operates unless all behaviors stop functioning at the same time.

While the blackboard architecture has a number of attractive features, difficulties are

Table 1. Variety of Multi-agent Systems (Corkill 2003)

	Form	Description
1	*Directly Interacting Agents*	Agents directly communicate with each other (or broadcast to everyone) and need to decide which information to share in the process of problem solving.
2	*Agents with Blackboard Agent*	Agents can interact indirectly with each other via the blackboard and need to make decisions locally on what they should be doing.
3	*Agents with Blackboard and Control (Manager) Agent*	A typical configuration of the blackboard architecture in which the control agent tells the other agents what to do next. This agent is simply an extension of **Form 2**.
4	*Full-Fledged Blackboard Agents*	Multiple blackboard systems are affiliated on one big multi-agent system. This is equivalent to **Form 1** in which each agent gets replaced by the agent in **Form 3**.

still present and agent-based systems, especially multi-agent systems, are instead winning a vote as a revolutionary method in controlling an autonomous robot. A number of multi-agent control systems are found in the recent AI literature (Soler et al. 2000; Sierra, L´opez de M`antaras, and Busquets 2001). These systems are basically an extended form of the blackboard system because of the fact that multi-agent systems in a way share some characteristics with blackboard systems. For example, a multi-agent system has a collection of agents (also called knowledge sources (KSs) in a blackboard system) which collaborates in problem solving forming the "cooperating expert". In fact, Corkill (2003) suggested a blackboard system could be seen as a variation of multi-agent systems (Table 1). However, in contrast to blackboard systems, multi-agent systems normally emphasize the following attributes of a control strategy: *distribution* (no central data repository), *autonomy* (local control), *interaction* (communication and representation), *coordination* (achieving coherence in local control decisions), and *organization* (emergent organizational behavior).

The goal of this thesis is to design and implement a naive but robust and easily expandable robot control package that is portable to heterogeneous system and hardware platforms, starting with the commercial robot kit as a test bed. Having said that, the system takes advantages of a multi-agent blackboard and a little bit of behavior-based approaches to construct the robot control system. Figure 7 depicts the simplified diagram representing the multi-agent system using a blackboard. The system in fact takes the second form of the above table (*Agents with Blackboard Agent*). The agents basically interact with the other components of the system by manipulating information on the blackboard. The blackboard mainly operates as a central repository for all shared

information and a communication medium for all agents. The information on the blackboard may represent facts, assumptions, and deductions made by the system during the course

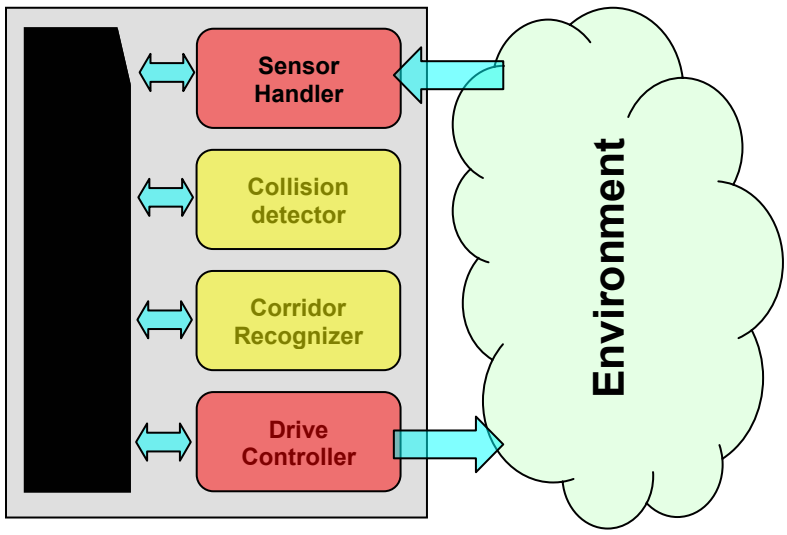


Figure 7. Multi-agent System with Blackboard

of solving a problem. An agent is a partial problem solver which may employ a different problem-solving strategy and try to contribute to the solution by viewing the information on the blackboard. The system has four independent agents such as *Fuzzy Collision Detector*, *Corridor Recognizer*, *Sensor Handler*, and *Drive Controller*. Note that the arrows in Figure 7 (above) represent information flow. Figure 7 shows that all four agents are allowed to read / write information on the blackboard. Each one of the four agents basically executes their tasks independently using information on the blackboard and posts any result back to the blackboard.

While employing a blackboard as the global database, the system differs from the blackboard architecture by having no control component with an inference mechanism. This loss actually benefits our system by allowing agents to make dynamic decisions locally. Then, a question arises. Is the blackboard merely a medium to share information? The blackboard surely functions as a communication board; so the answer is partially "yes" but there is more to it. In fact, agents do not need to communicate with each other

for information due to the existence of the blackboard. The blackboard assures the independence for each agent and maintains decentralization for the system, which ultimately results in allowing the system to have parallel and distributed intelligence.

It is importantly noted that the robot control system was originally inspired by Brooks' subsumption architecture (Brooks 1986), but the system still does not allow a purely reactive behavior. That is, the incoming sensory information must always go through the blackboard before causing any physical actions. This hybrid system, however, can increase the performance of real-time sensitivity over that of the blackboard architecture by localizing the decision-making process and maintaining decentralization. The following sections extensively explain how each agent is implemented and incorporated with the whole system.

4.2 AGENTS

The four agents (*Sensor Handler, Collision Detector, Corridor Recognizer, and Drive Controller*) and the blackboard make up the control system. The agents are classified into two groups. The Sensor Handler and the Drive Controller belong to the first group that has access to and interacts with the environment (see Figure 7). The other two, the Fuzzy Collision Detector and the Corridor Recognizer are strictly prohibited from having direct access to the environment. As a result, they perform tasks based on the information acquired from the blackboard. There is no global controller for

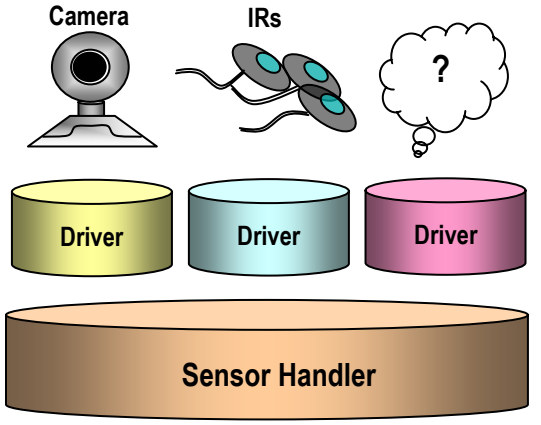


Figure 8. Sensor Handling Apparatus

these agents, and each of them independently tries to make a contribution to the system during a course of navigation.

The agents that directly interact with the environment are designed for the purpose of incrementally adding more tools to the system. Particularly, the Sensor Handler is responsible for all the sensors installed on the robot and should have such ability without making extensive efforts on the system modification and configuration. As we discussed in Chapter 2, the agents are composed of two layers, the Hardware Layer and the Component Layer. The Sensor Handler specially benefits from this layered architecture by having device drivers at the Hardware Layer (Figure 8). Layering the drivers between the Component Layer and the physical sensor apparatus, the Sensor Handler can maintain its adaptability by having task-oriented modules at the Component Layer which only deal with the symbolic representation of sensor data (e.g. digitized frequencies or an array of pixels). Each sensor requires a driver written in a native programming language such as C or C++ which runs on the operating system (e.g. **.exe* for Windows). Although the drivers can be freely designed and implemented, they are required to receive input commands and return output character strings by the Sensor Handler (Table 2). This standardized I/O specification facilitates the implementation process of the agent.

Table 2. Example of Driver I/O

Driver: Infrared.exe

Input	Command
-1	***Halt***
0	***Idle***
1	*Raw data*
2	*Distance*

Output (Character string)
[data ID] *[sensor 1] [value]* *[sensor 2] [value]* *... [sensor n] [value]*

*Bolded inputs and commands in the input table are reserved by the Sensor Handler.

The reasons of having the sensor-handling agent in the robot control system are the

following: *organization*, *enhancement,* and *perception*. During system initialization, the Sensor Handler looks for sensors and determines what is available and what is not because the other agents are heavily dependent on the sensor readings, and thus knowing the availability in advance minimizes errors by deactivating the agent with no sensor inputs. Although some sensors are used specifically for a particular agent, this agent has no authority to fully manipulate those sensors. The agent that communicates with the environment indirectly through the blackboard needs no access to sensors to interact directly with the outside world. This paradigm also simplifies the design phase of introducing a new agent to the system because we only need to consider the agent's behavior assuming the sensor data is already posted on the blackboard. The layered architecture surely empowers the system with respect to the sensor organization, modulation and enhancement.

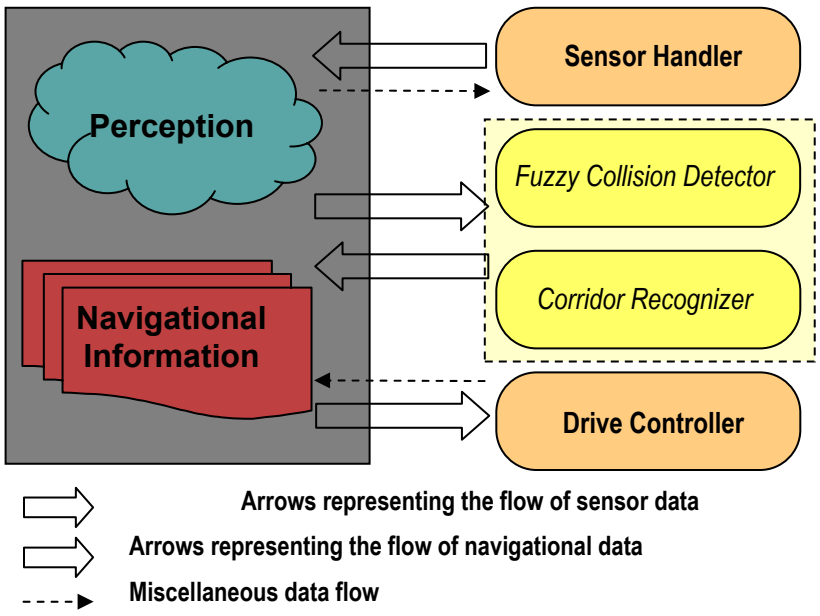


Figure 9. Data Flow between Blackboard and Agents

Another important mission of the Sensor Handler is to maintain the up-to-date perception about the surrounding environment so as to have the latest information available for the other agents. The current system handles the following sensors, a camera and nine IR sensors. The camera updates the front view of the surrounding environment

capturing approximately ten frames per second. The IR sensors measure surrounding objects almost continuously. The camera and the IRs mostly operate independently in a separate thread, but sometimes synchronized in order to properly update the blackboard. Figure 9 shows how the agents interact with the blackboard and manipulate the information. The agents in the dotted rectangle such as Fuzzy Collision Detector and Corridor Recognizer are the ones that use fresh data (perception) acquired via the Sensor Handler, and they in turn update the information about the navigation status on the blackboard.

The Drive Controller is also the one that has access to the environment. The agent primarily holds responsibility for the robot's actuator via the device driver that controls motors through the stepper control module. The Drive Controller monitors the blackboard and uses the navigational information for locomotion. With the layered framework, the Drive Controller has the same advantage as the Sensor Handler in its simple and structured implementation. For example, the agent is made of modules responsible for the motor initialization and termination, the communication between layers, and the maneuvering of the robot. A decent number of motion parameters such as velocity, acceleration, turn-angle, straight distance, and driving duration are arranged for achieving flexible movements.

4.3 CORRIDOR RECOGNITION

During a course of actions taken by the robot, a smooth and successful navigation is directly dependent on how well the robot recognizes lines representing a corridor. Extracting a path (or a road) from an image is one of the popular problems in the field of

machine vision. Although in the past researchers have applied various sensors to solve this problem, vision has proven to be the most successful because of the high information content and the sensor passiveness (McDonald et al. 2001). In order to steer a path through the environment, the vehicle must secure the middle space in a hallway. Most view-based approaches use optical flow to approximate the distance, angle, or whatever information necessary to drive the robot in a corridor. Roughly, the typical steps used in most techniques are the following.

1. *Image segmentation (e.g. regions, edges, intensity characteristics)*
2. *Feature selection and extraction (e.g. line, corner, shape)*
3. *Pattern recognition (e.g. objects, lanes, hallways)*

The first step normally involves low-level image processing techniques. Pixel-based operations (e.g. threshold and histogram operators) morphological analysis (e.g. thinning and skeletonization), and digital filters (e.g. noise reduction and other enhancement filters) are often used in this step. After shaping or simplifying the image, we want to find features in the image necessary to identify the topological information. Feature detectors (e.g. edge detectors and other feature detectors) are most often utilized to extract the interesting patterns in the image, and the features get sometimes transformed into another plane (e.g. Fourier, Hough and other transforms) and exclusively selected. In order to identify the meaningful entities in the input image, we sometimes need to provide background knowledge about things we are interested in. The third step involves heuristic techniques such as fuzzy logic to classify the road shapes (Shanahan et al. 1999), and neural networks with machine learning capability (Jochem, Pomerleau and Thorpe 1995). These systems are precise and adaptive to unseen environments, but require a fairly large

amount of a-priori knowledge and fine tuned parameter settings.

In the system used for the experiment, the corridor recognition agent roughly consists of two levels of image processing modules. The low-level image processing basically involves tasks specified at step one, image segmentation. Figure 10 shows each step of the segmentation process. The JPEG image is acquired from the camera at the resolution

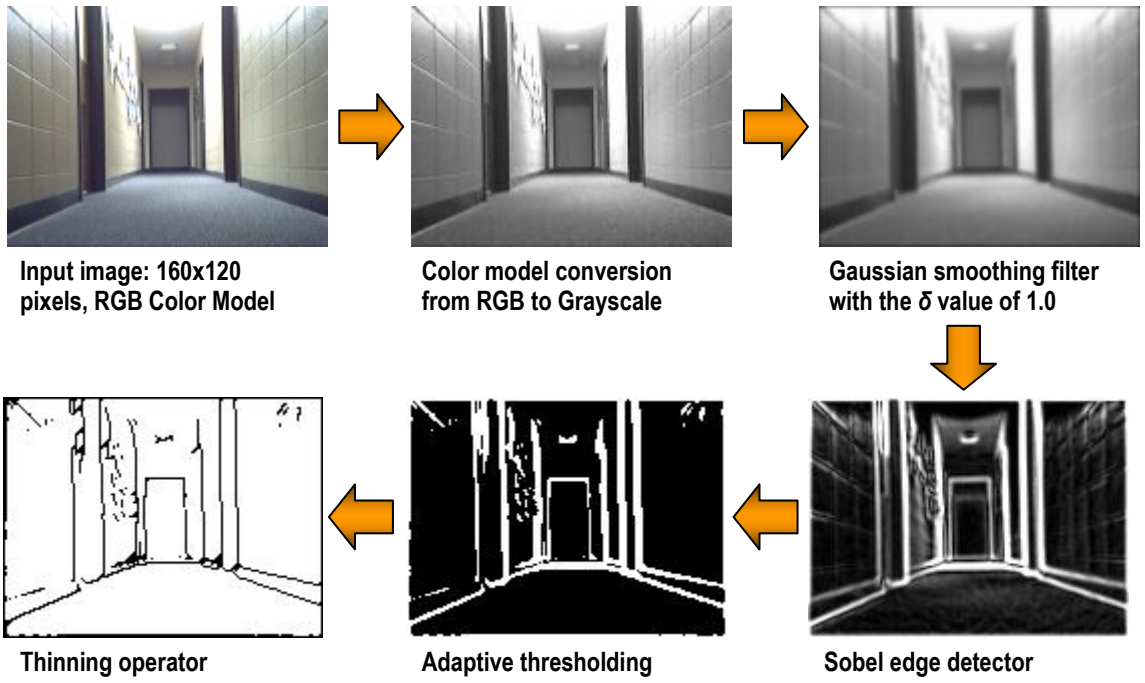


Figure 10. Flow of Low-level Image Processing

of 160 x 120 pixels with a 32-bit (ARGB) color model. The image is then converted to grayscale (8-bit) for ease of computation while the pixel intensity values are stored in an integer array. Applying a *Gaussian filter* reduces noise in the image by blurring neighboring pixels and helps the edge detector to select correct edges. A *Sobel edge detector* is applied on the smoothed image. The grayscale colors in the image are reduced to black and white using the *adaptive thresholding* operator so as to remove unwanted details before applying a thinning operation. The threshold value is dynamically selected by performing a statistical analysis on the sampled pixel intensity values. Because of the

nature of the Sobel operator, the *thinning operator* must be applied to reduce the lines with several pixels width to a single pixel width. In contrast to the lower-level image processing, a variety of research has been conducted on the feature extraction steps. For example, Broggi and Bertè (1995) identified the road comparing the pre-encoded synthetic road models with the road scene acquired from a camera. McDonald et al. (2001) used a Hough transform to detect roads during motorway driving scenarios.

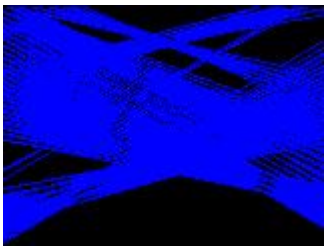
The final result of applying the Hough transform after the feature extraction process.

Figure 11. Extracted Corridor

In this experiment, the Hough transform with a-priori knowledge (constraints on the geometry features of a corridor) is used for extracting the line segments of a corridor path (Figure 11). The Hough transform was invented in 1962 by P.V.C. Hough and has been a widely accepted engineering technique in various applications. The Hough transform is simply a parameter estimation that uses a voting mechanism. Each point on a line (or a curve) votes for several combinations of line parameters (Jain, Kasturi and Schunck 1995). In principle, a Hough transform can detect arbitrary shapes in images, given a parameterized description of the shape in question. The implementation of the Hough transform for line detection uses a 2-D array for accumulating the voted points that represent parameterized space (Whelan and Molloy 2000). Figure 12a shows the points voted for by the Hough transform in a parameterized plane in which the linear equation is inversed so that the variables become constants and the constants are variables of interest. Figure 12b is the result of inversing the parameterized plane into an image plane. The points transformed back on to the image plane constitute straight lines. In fact, the Hough transform possibly

a. **An image of the parameterized space plotting the points voted by Hough transform.**

b. **The inverse of the parameterized space. Straight lines are the possible candidates for the corridor path.**



c. **Two winners out of hundreds of corridor candidates. Constraints on the hallway geometry knocked off the incorrect candidates.**

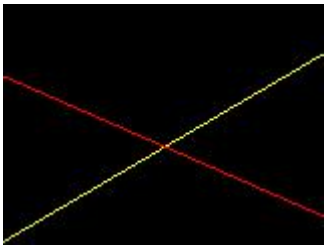
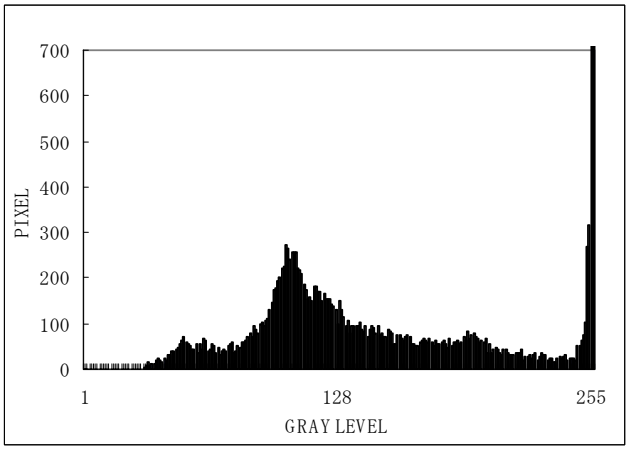


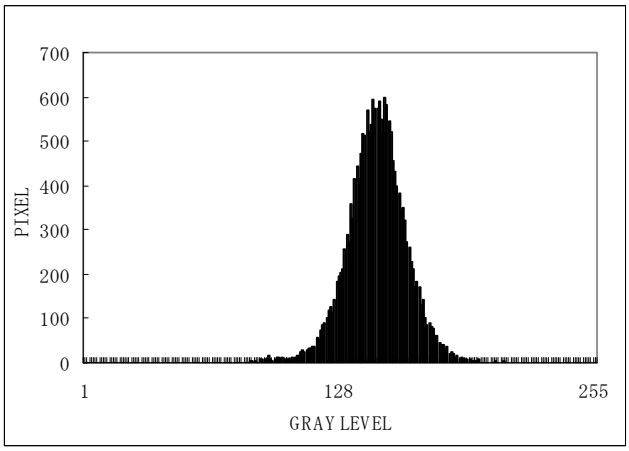
Figure 12. Hough Transform and Line Selection

extracts all line segments in the image. In order to select lines which best represent the hallway, we need to use knowledge about corridors. The selection process involves two steps, *selection* and *verification*. In the selection phase, the lines whose slope does not fit the geometry constraint are thrown out first. Next, each line is compared with the edge maps (the final image in Figure 10). In this step, the pixels of a line matching the corresponding edge points are counted, and only lines with the matching pixels that go above a certain threshold are selected. At the verification step, after selecting the corridor path, the corridor recognition agent double-checks the lines to see if they really represent the hallway or not by performing a complete histogram analysis. The typical patterns of the histograms of images representing the environment (corridors, walls, and objects) are shown in Figure 13. Each histogram to some extent exhibits significant characteristics of the image representing the target situation. The first histogram (Figure 13a) is a typical intensity distribution for the image that faces straight along a hallway (as in Figure 11). The pixels spread over all intensity levels fairly evenly, and the moderate peaks represent the floor between the boundaries of a corridor. The next

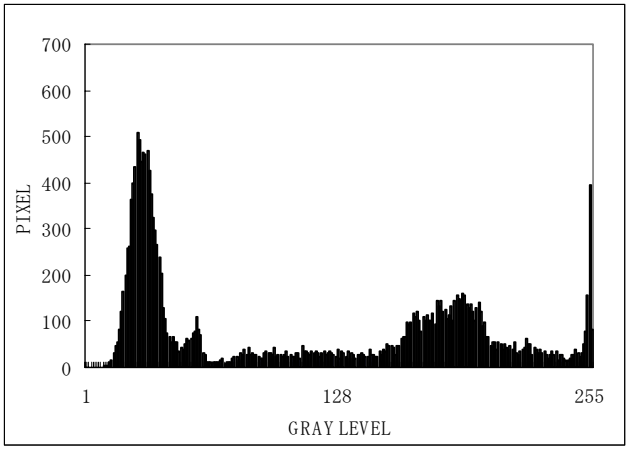
graph (Figure 13b) demonstrates massive pixel clusters lying within the short range of intensity levels, which suggests that the image contains a large and somewhat homogeneously colored object. As a matter of fact, the image actually contains one big homogeneous object, the white wall. The corridor detection agent is most likely to reject the image because of the extreme intensity characteristics. The last histogram (Figure 13c) shows again a moderate intensity distribution, but the pixels are apt to belong to the tips of gray-levels. As a matter of fact, this is an example of obstacles. In this particular example, the camera is facing closely a large and colored object in the robot's path. It is difficult to make assumptions about the intensity of obstacles since any object can be a possible candidate for being an obstacle no matter how big it is, which color it is, or what shape it has. However, the chances that



a. The robot (camera) is facing a narrow corridor



b. The robot is facing a wall



c. The robot is facing an object in the path

Figure 13. Intensity Histogram Samples

the obstacle intensity pattern fits the corridor intensity pattern may not probably be so frequent.

4.4 FUZZY-BASED COLLISION AVOIDANCE

Fuzzy logic has been widely accepted in mobile robot navigation because of a number of advantages. First, fuzzy logic controllers can easily incorporate heuristic knowledge in the form of if-then rules manipulating the symbolic representation of an environment. Secondly, robot navigation in an unstructured or unseen environment mostly requires a non-linear, dynamic and fast system to map sensor values to the robot actions. Also, fuzzy logic controllers have shown a certain degree of robustness in terms of a variability and uncertainty in the parameters (Saffiotti 1997).

While fuzzy logic has been applied to many aspects of robot navigation, collision avoidance and obstacle avoidance are the most popular territories and one of the major research areas in mobile robotics. The robot capable of navigating autonomously in an unstructured environment must know the ways to keep it safe during the course of navigation. Numerous research projects concerned with collision avoidance use a fuzzy logic controller to approximate reasoning necessary for dealing with uncertainty in combination with various sensors. For example, Martinez et al. (1994) applied a fuzzy logic controller using sonar sensors as range finders and IR sensors as proximity detectors in order to estimate proximity, distance to an object, speed, and direction. Tunstel, E. and Jamshidi (1994) employed a fuzzy-based mechanism to realize wall-following behavior using four optical range sensors. Wijesoma (2001) implemented a fuzzy navigation system coupling IR proximity detectors and laser scanners. Cho and

Nam (2000) made a fuzzy controller to steer a robot using image inputs.

The agent called *Fuzzy Collision Detector* is a fuzzy-based collision avoidance controller responsible for the safety of the robot used in this experiment. There are roughly three steps in applying a fuzzy logic controller. The first step is the input fuzzification in which the crisp input values are fed into the antecedent membership functions, which maps to appropriate linguistic terms. The second step is to draw inferences by matching rules. Note that the input (antecedent) typically matches more than one rule, and all the conclusions of the matched rules are averaged out with a Boolean operation such as union. Lastly, the output is defuzzified and yields crisp numerical values which promise safety during navigation. Figure 14 depicts an example of the complete flow of fuzzy inference mechanism (although the inference processes may repeat).

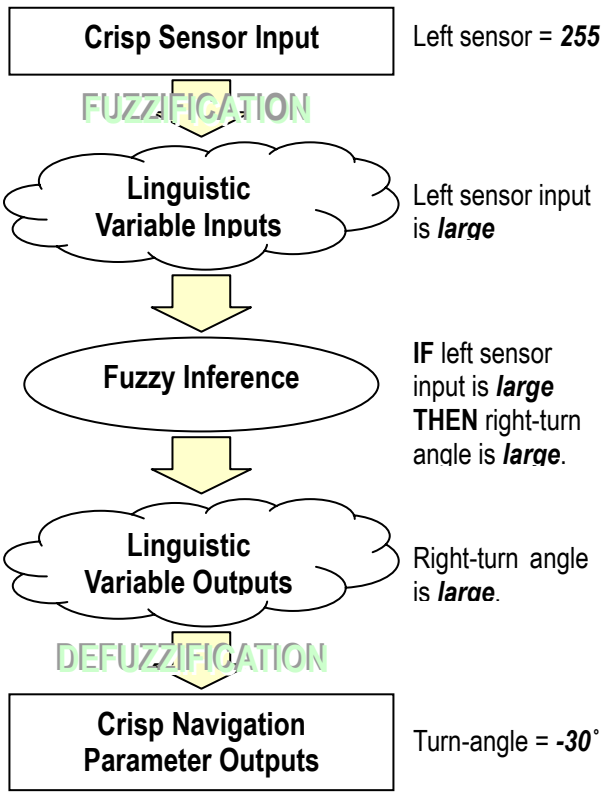


Figure 14. Example of Fuzzy Inference

The fuzzy logic controller has one input fuzzy set for *sensor value* and three output fuzzy sets such as *linear-distance*, *velocity* and *turn-angle*. Each set is defined by one or more membership functions that map numeric values onto linguistic terms. The membership functions of each fuzzy set except the turn-angle fuzzy set are shown in

Figure 15. The fuzzy-based agent is fed with sensor values as an input, acquired from a set of infrared proximity detectors. The values are fuzzified with designated linguistic terms (*near*, *medium*, and *far*). Among three output fuzzy sets, the turn-angle fuzzy set has been uniquely defined. The angle lies between -30˚ and 30˚ as a default (adjustable via the navigation software interface). The total angle (60˚ in this case) divided into six amplitudes is represented by six member functions, and each of which is associated with the following linguistic terms.

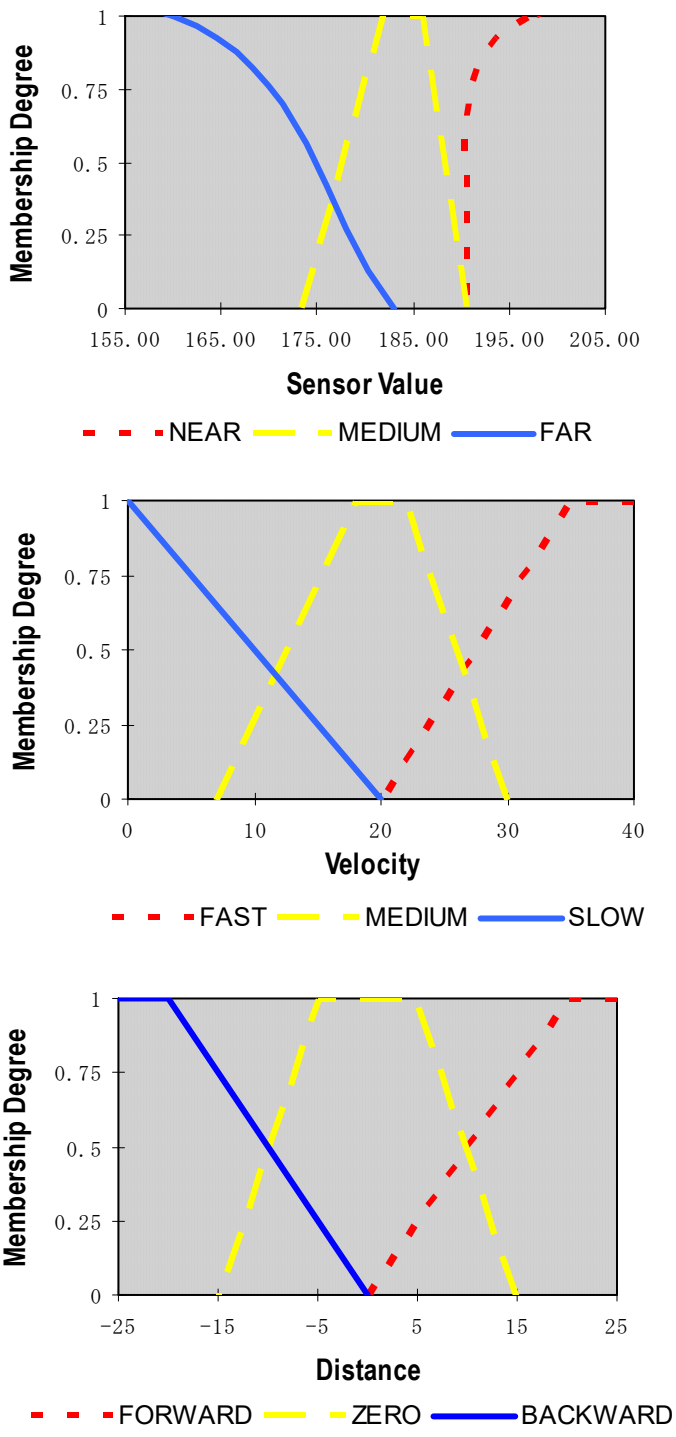


Figure 15. Membership Functions of Fuzzy Sets

- *positive-left (PL)*
- *positive-center (PC)*
- *positive-right (PR)*
- *negative-left (NL)*
- *negative-center (NC)*
- *negative-right (NR)*

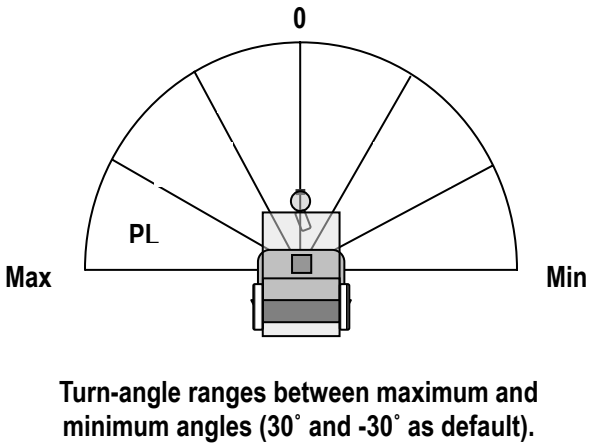


Figure 16. Turn-angle Fuzzy Set

Figure 16 projects the six linguistic terms on the partitioned turn-angle fuzzy set. This scheme has already been established and proven to be an effective method (Fayad and Webb 1999). Once the input is fuzzified and all the output fuzzy sets are defined as appropriate linguistic terms, the fuzzy inference engine looks for a match between the input and the outputs.

The agent has 17 fuzzy rules in total: seven rules for the front sensors, two for the rear sensors, and four for each of the left and right sensors as shown in Table 3. Left columns in the table correspond the antecedents of the fuzzy rules. The nine IR sensors are used as inputs, three sensors in the front, two in the back, and two on each side. The sensor arrangement is shown in Figure 17. Right columns in the table are the sets of conclusions. Notice that the rules do not necessarily carry the same conclusions.

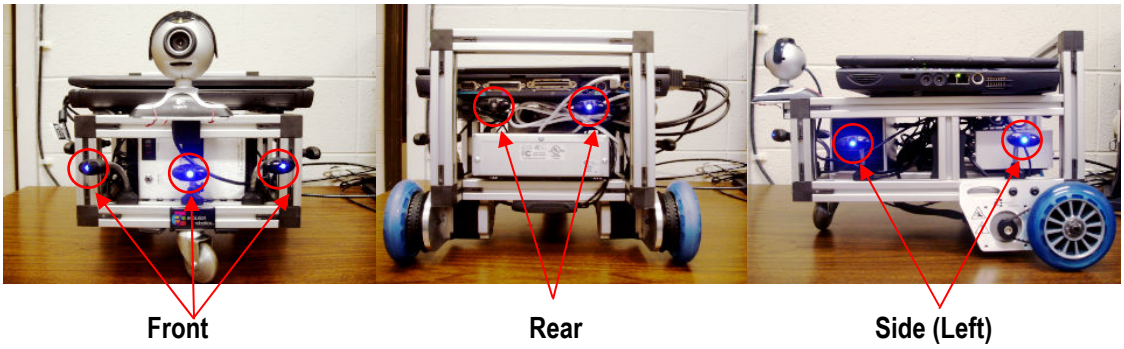


Figure 17. IR Sensor Arrangements

One may wonder if the rules sufficiently cover all possible collision circumstances which may demand more than seventeen rules. As a matter of fact, the number of inferences we can draw from these 17 rules is at most 192 combinations, and the

linguistic output explosively multiplies on to numerical values in the defuzzification process. Defuzzification is the last step of the fuzzy logic controller. The outputs from each rule firing are combined using a fuzzy union operator, and the crisp output value is defuzzified by computing a centroid of the area that is enclosed by the output member functions.

Table 3. Rule Set for Fuzzy Logic Controller

	Antecedents (IF)	Conclusions (THEN)
1	[Front Center] sensor is NEAR	[Distance] is BACKWARD [Velocity] is MEDIUM
2	[Front Center] sensor is MEDIUM	[Distance] is ZERO [Velocity] is SLOW
3	[Front Center] sensor is FAR	[Distance] is FORWARD [Velocity] is FAST
4	[Front Left] sensor is NEAR	[Distance] is BACKWARD [Velocity] is MEDIUM
5	[Front Left] sensor is MEDIUM	[Turn Angle] is NL
6	[Front Right] sensor is NEAR	[Distance] is BACKWARD [Velocity] is MEDIUM
7	[Front Right] sensor is MEDIUM	[Turn Angle] is PR
8	[Rear Left] sensor is NEAR	[Distance] is ZERO [Velocity] is SLOW
9	[Rear Right] sensor is NEAR	[Distance] is ZERO [Velocity] is SLOW
10	[Left Front] sensor is NEAR [Left Rear] sensor is NEAR	[Turn Angle] is NL
11	[Left Front] sensor is NEAR [Left Rear] sensor is MEDIUM	[Turn Angle] is NC
12	[Left Front] sensor is NEAR [Left Rear] sensor is FAR	[Turn Angle] is NR
13	[Left Front] sensor is MEDIUM [Left Rear] sensor is FAR	[Turn Angle] is NL
14	[Right Front] sensor is NEAR [Right Rear] sensor is NEAR	[Turn Angle] is PR
15	[Right Front] sensor is NEAR [Right Rear] sensor is MEDIUM	[Turn Angle] is PC
16	[Right Front] sensor is NEAR [Right Rear] sensor is FAR	[Turn Angle] is PL
17	[Right Front] sensor is MEDIUM [Right Rear] sensor is FAR	[Turn Angle] is PR

Variables are enclosed by [ ] and linguistic terms are capitalized. Rules are constructed based on the membership functions given in Figure 15 except for the turn-angle fuzzy set that is explained in Figure 16.

5. EXPERIMENTS

5.1 EXPERIMENTAL SETUP

The experiments are conducted on a narrow straight corridor in an office-like indoor environment with a relatively dimmed lighting condition. The corridor extends about 100 feet in length and 4 feet in width. In a narrow opening like a corridor, the robot moves with a speed slower than an average walking speed. The agents *Drive Controller*, *Sensor Handler* and *Blackboard* are the essential components that must be executed during navigation. The collision detection agent is independently executable, but the corridor recognition only makes high-level decisions and must corporate with the agent that makes decisions about motor control parameters.

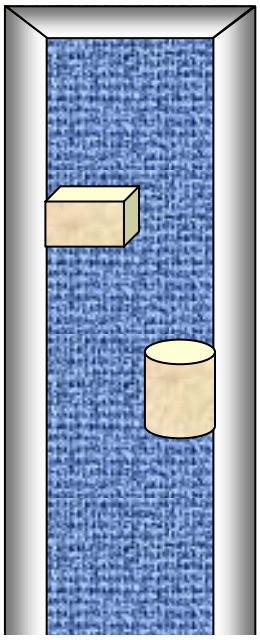


Figure 18. Partial Environment Setup

The most important mission of the experiments is to analyze and verify the performance of the agents and the control system as well as to accomplish a successful navigation. Therefore, the robot is evaluated in each of the criteria listed as the following.

1. Agents (targets: *Fuzzy Collision Detector* and *Corridor Recognizer*)
    a. Robot with collision detection
    b. Full feature (collision detection and corridor recognition)
2. Robot control system
    a. Modularity and usability
    b. Safety (system safety and robustness)

For the purpose of evaluating the agents' performance with respect to safety, on each experiment the robot runs in a corridor with obstacles as in Figure 18. Because of the dead-end in the corridor, an extra control subroutine is added to the *Drive Controller* in which the robot turns around if the corridor recognition agent recognizes that the robot is facing a wall. There are some more interesting situations such as a doorway and a corner that have been tested to see how the robot tackles to these problems.

5.2 EXPERIMENTAL RESULTS

As a result, the robot has shown both desired and problematic behaviors. In the matter of collision detection, the robot is able to avoid collisions with obstacles and walls. The fuzzy-based collision detection agent maneuvered the vehicle around and navigated it to the end of the hallway without collisions. There were also some problems, however. First of all, the agent was often distracted by ambient light, which caused the retardation in navigating the robot. As a possible solution, applying additional filters (e.g. low-pass, high-pass, or band-pass filter) to protect the sensor input (although most IR sensors are already filtered) from stray light may improve sensor readings before applying collision detection. However, oftentimes it will require a sensor calibration in each and every unique environment.

The second problem is the advisability of fuzzy rules. Although the rules are, to some extent, optimized as the result of repetition of trial and error, the agent is sometimes making magnified conclusions about the turning angle in large amplitude, which results in zigzag locomotion. Relying only on the collision detection agent that only employs infrared sensors may not be able to fix the problem completely, but there is still plenty of

room for improving the sensor arrangement and the corresponding fuzzy rules. In addition, using more sensors to cover broader ranges is definitely a plus.

The corridor recognition agent analyzes the visual input and the contents of the front view. As a result, most of the time, the agent has made correct decisions on recognizing a corridor, but they are not perfect enough in terms of safety in navigation. Although an improvement still needs to be made with respect to the accuracy in selecting a correct set of lines which represent a corridor, the main problem is the way to handle the shared knowledge on the blackboard. In principle, the robot control system has no central brain in the system, and any information posted on the blackboard must be handled and interpreted by each agent. In the current system, the *Drive Controller* is the only agent dealing with the shared information that reflects on the robot's actuators.

A subtle timing difference in the results of agents sometimes causes destructive behaviors. As a matter of fact, the robot has often stopped and backed up mistakenly and sometimes even hit the walls during the experiment when executing both the collision detection and corridor recognition agents. There are two reasons for this malfunctioned behavior. One reason is a failure in *knowledge synchronization*. Because of the agents that are completely independent and perform tasks in parallel, one agent occasionally delays in updating the blackboard. The delays are most likely instantaneous and unnoticeable but sometimes come in between two states where the robot is making a significant transition. For example, the robot was facing along the corridor and then turned to the wall in the next ten milliseconds. At this moment, the blackboard could contain contradictory facts about the environment such as "going straight with a moderate speed" (the message from the collision detection agent) and "facing a wall" (the message

from the corridor recognition agent). There are options in solving this problematic situation. One option is that either the *Drive Controller* must optimally resolve this inconsistency, or the blackboard must know how to synchronize the update timing. The other option is to introduce a new agent that resolves the conflict and synchronizes the blackboard information.

The other possible reason is how to handle false claims given by the agents, particularly the corridor recognition agent. A sequence of images obtained during operation of the corridor recognition agent is given in Appendix A. The agent classifies three things, a corridor, walls, and obstacles. The images in the appendix show that the agent identifies the corridor almost perfectly. However, the ratio of correctness drops in identifying the walls and the obstacles. In fact, it is extremely difficult to always make correct judgments in the dynamic scene without giving an appropriate amount of hints (knowledge about a particular environment and obstacles which reside in the environment) to the agent. It is the simplest solution with 2 obdawblac;w thatils, ddring

originally designed to do only two things: avoid collisions and recognize a hallway. However, the robot also demonstrated unexpected movements: obstacle avoidance and doorway navigation. Obstacle avoidance has emerged due to the coupling of the agent's behavior and the environment (the corridor wall). Figure 19 illustrates the obstacle avoidance behavior exhibited by the robot. The robot steered around the rectangular obstacle during navigation because of the wall that kept the robot from going away. The agent would behave differently if the agent were situated in the wide open space. The same principle applied to doorway navigation. The doorway navigation basically does not differ from collision detection based on the assumption that the door is open. However, the robot has to deal with more variety than a common hallway situation in topological features such as a doorframe. The robot in fact went through the doorway in an office-like environment without collision although it did so moving zigzag as expected.

As mentioned in the previous chapters, the objective of this experiment was to design and build a robot control program that is independent of the system platforms and easy to expand (modify) for future study. To begin with, the control system succeeded in facilitating modularity and usability. The complete modulation in the multi-agent architecture brought forth an effortless implementation, and the intelligible user interface has navigational parameters all adjustable and realizes the smooth experiment processes. The GUI (Graphical User Interface) written in Java is shown in Appendix B.

As opposed to the modulation, the program also left some problems regarding the system stability. During navigation, the control system often lost control because of a system failure. This could be caused by many reasons; nonetheless, it is mainly caused by memory consumption. Although the agents written in Java already have an ability of

automatic garbage collection, the device drivers written in C++.NET are not capable of doing it on their own and need to be provided with one. Meanwhile, in order to recycle resources efficiently, those drivers must be designed with extra caution considering all possible cases for memory usage.

5.3 DISCUSSION

The vision-based agent, the fuzzy-based agent, and the agents responsible for hardware components all cooperate within the blackboard-based multi-agent system. The robot and the control system were presented and analyzed in the experiments. As a result, the robot did not exhibit the perfectly desired performance, but the multi-agent approach in the design criteria has proved its feasibility in mobile robot navigation. The problems faced during the experiments are more related to the calibration against an environment and the parameter adjustment on the agents than the fundamental design criteria of the control system. The proposed layered architecture enabled the control system to be easily expandable, and the use of Java technology made the system independent of operating systems and robot hardware.

A fuzzy logic controller is integrated into the collision detection agent. Solely with this agent, the robot demonstrated safe navigation in a hallway using a set of IR proximity sensors. The unreliable sensor input is compensated for by the fuzzy logic controller, and the agent made moderate decisions in the experiments. The corridor recognition agent processes an incoming image to determine lines representing a hallway. The agent has not yet performed at the level of satisfying the safety criterion while detecting corridors more than ninety percent of the time. Future improvement is

necessary for optimizing the recognition tasks. However, when the above two agents collaborate on navigation, the real problem emerged and more work is required to fix the problem. Because of the absence of a control mechanism, the system suffers serious problems regarding information sharing. As the current ongoing research, possible solutions are being implemented to compensate for this problem. For example, the module was added to the blackboard agent which synchronizes the results given by the other agents. Also, the next step is to introduce a new agent that accommodates all the information acquired from agents and schedules (or prioritizes) the robot actions by evaluating information on the blackboard. Further study is sought to design an agent which actually performs the landmark-based navigation extending machine vision techniques, and also an agent with a neuro-fuzzy controller for learning an environment so that no manual calibration is necessary.

REFERENCES

Arras, K.O. and Tomatis, N. (1999) Improving robustness and precision in mobile robot localization by using laser range finding and monocular vision. *Proceedings of the Third European Workshop on Advanced Mobile Robots*, Eurobot'99, Zurich, Switzerland. http://citeseer.nj.nec.com/arras01multisensor.html

Arras, K. O., Tomatis, N. and Siegwart, R. (2000) Multisensor on-the-fly localization using laser and vision. *Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Takamatsu, Japan, 793-8.

Argyros, A. and et al. (2002) Semi-autonomous navigation of a robotic wheelchair. *Journal of Intelligent and Robotic Systems* 34: 315-29.

Asensio, J. R., Martínez, J. M. and Montano, L. (1998) Navigation among obstacles by the cooperation of trinocular stereo vision system and laser rangefinder. *3rd. IFAC Symposium on Intelligent Autonomous Vehicles*, IAV'98, Madrid, Spain, 456-61.

Benet, G. and et al. (2002) Using infrared sensors for distance measurement in mobile robots. *Robotics and Autonomous Systems* 40: 255-66.

Bertozzi, M., Broggi, A. and Fascioli, A. (2000) Vision-based intelligent vehicles: state of the art and perspectives. *Robotics and Autonomous Systems* 32: 1-16.

Borenstein, J. and Koren, Y. (1988) Obstacle avoidance with ultrasonic sensors. *IEEE Journal of Robotics and Automation* RA-4(2): 213-8.

Borenstein, J. and Koren, Y. (1991) The Vector Field Histogram -- Fast obstacle-avoidance for mobile robots. *IEEE Journal of Robotics and Automation* 7(3):

278-88.

Broggi, A. and Bertè, S. (1995) Vision-based road detection in automotive systems: a real-time expectation-driven approach. *Journal of Artificial Intelligence Research* 3: 325-48.

Brooks, R. A. (1986) A robust layered control system for a Mobile Robot. *IEEE Journal of Robotics and Automation*, RA-2(1): 14-23.

Brooks, R. A. (1991) Intelligent without representation. *Artificial Intelligence* 47: 139-59.

Cho, J. T. and Nam, B. H. (2000) A study on the fuzzy control navigation and the obstacle avoidance of mobile robot using camera. *Proceedings of the 2000 IEEE International Conference on System, Man and Cybernetics*, Nashville, TN, 2993-7.

Corkill, D. D. (1991) Blackboard systems. *AI Expert* 6(9): 40-7.

Corkill, D. D. (2003) Collaborating software: blackboard and multi-agent systems & the future. *Proceedings of the International Lisp Conference 2003*, New York, NY. http://dancorkill.home.comcast.net/pubs/ilc03.pdf

Crisman, J. and Cleary, M. (1998) Progress on the deictically controlled wheelchair. In Mittal et al. eds., *Assistive technology and AI*. LNAI-1458. Berlin: Springer-Verlag, 137-49.

Fayad, C. and Webb, P. (1999) Optimized Fuzzy Logic Based on Algorithm for a Mobile Robot Collision Avoidance in an Unknown Environment. *Proceedings of the 7th European Congress on Intelligent Techniques and Soft Computing*, Aachen, Germany. http://www.cs.nott.ac.uk/~cxf/Papers/Optimised_Fuzzy_Logic.pdf

Fod, A., Howard, A. and Mataric´, M. J. (2002) A Laser-based people tracker.

Proceedings of the 2002 International Conference on Robotics and Automation, Washington DC, 3024-9.

Goldberg, S.B., Maimone M. W. and Matthies L. (2002) Stereo vision and rover navigation software for planetary exploration. *Proceedings of the 2002 IEEE Aerospace Conference*, Big Sky, MT, 2025-36.

Gomi, T. and Griffith, A. (1998) Developing intelligent wheelchairs for the handicapped. In Mittal et al. eds., *Assistive technology and AI*. LNAI-1458, Berlin: Springer-Verlag, 150-78.

Hirschmüller, H. (2002) Real-time correlation-based stereo vision with reduced border errors. *International Journal of Computer Vision*, 47(1/2/3): 229-46.

Hundelshausen, F. v., Behnke, S., and Rojas. R. (2002) An omnidirectional vision system that finds and tracks color edges and blobs. In Birk, A., Coradeschi, S., and Tadokoro, S., eds., *RoboCup 2001: robot soccer world cup V* ,LNAI-2377, 103-22. Berlin: Springer-Verlag.

Jain R., Kasturi, R. and Schunck, B.G. (1995) *Machine vision*. New York: McGraw-Hill.

Jensfelt, P. (2001) Approaches to mobile robot localization in indoor environments. Ph.D. dissertation, Department of Signals, Sensors and Systems, Royal Institute of Technology, Stockholm, Sweden.

Jochem, T., Pomerleau, D. and Thorpe C. (1995) Vision guided lane transition. *Proceedings of the 1995 IEEE Symposium on Intelligent Vehicles*, Detroit, MI, 30-5.

Katevas, N. I. and et al. (1997) The autonomous mobile robot SENARIO: a sensor-aided intelligent navigation system for powered wheelchairs. *IEEE Robotics and Automation Magazine* 4(4): 60-70.

Kimiaghalam, B. and et al. (2001) A multi-layered fuzzy inference systems for autonomous robot navigation and obstacle avoidance. *Proceedings of the 10th IEEE International Conference on Fuzzy Systems*, Melbourne, Australia, 340-3.

Kube, C. R. (1996) A minimal infrared obstacle detection scheme. *The Robotics Practitioner: The Journal for Robot Builders* 2(2): 15-20.

Lankenau, A. and et al. (1998) Safety in robotics: the Bremen autonomous wheelchair. *Proceedings of the 5th International Workshop on Advanced Motion Control*, Portugal, Coimbra, 524-9.

Lankenau, A., Röfer, T. and Krieg-Bruckner, B. (2003) Self-localization in large-scale environments for the Bremen Autonomous Wheelchair. In Freksa and et al. eds., *Spatial Cognition III*. LNAI-2685. Berlin: Springer-Verlag, 34-61.

Levine, S.P. and et al. (1999) The NavChair Assistive Wheelchair Navigation System. *IEEE Transactions on Rehabilitation Engineering* 7(4): 443-51.

Liscano, R. and et al. (1995) Using a blackboard to integrate multiple activities and achieve strategic reasoning for mobile-robot navigation. *IEEE Expert* 10(2): 24-36.

Maaref, H. and Barret, C. (2002) Sensor-based navigation of a mobile robot in an indoor environment. *Robotics and Autonomous Systems* 38: 1-18.

Martinez, A., Tunstel, E. and Jamshidi M. (1994) Fuzzy logic based collision avoidance for a mobile robot. *Robotica* (12): 521-7.

Mataric´, M. J. (1992) Behavior-based control: main properties and implications. *Proceedings of the IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems*, Nice, France, 46-54.

Matsumoto Y. and et al. (1999) Exploration and map acquisition for view-based navigation in corridor environment. *Proceedings of the International Conference on Field and Service Robotics*, Pittsburgh, PA, 341-6.

Mazo, M. and et al. (2002) Experiences in assisted mobility: the SIAMO project. *Proceedings of the 2002 IEEE International Conference on Control Applications*, Anchorage, Alaska, 766-71.

McDonald, J. B., Franz, J. and Shorten, R. (2001) Application of the Hough transform to lane detection in motorway driving scenarios. *Proceedings of the Irish Signals and Systems Conference 2001*, Maynooth, Ireland. http://www.cs.may.ie/~johnmcd/publications/ISSC2001.ps.gz

Miller, D. (1998) Assistive robotics: an overview. In Mittal et al. eds., *Assistive technology and AI*. LNAI-1458. Berlin: Springer-Verlag, 126-136.

Nisbet, P. D. (2002) Who's intelligent? Wheelchair, driver or both? *Proceedings of the 2002 IEEE International Conference on Control Applications*, Anchorage, Alaska, 760-5.

Paletta, L., Frintrop, S. and Hertzberg, J. (2001) Robust localization using context in omnidirectional imaging. *Proceedings of the 2001 IEEE International Conference on Robotics and Automation*, Seoul, Korea, 2072-7.

Prassler, E., Scholz, J. and Fiorini, P. (1999) MAid: a robotic wheelchair roaming in a railway station. *Proceedings of the International Conference on Field and Service Robotics*, Pittsburgh, PA. http://voronoi.sbp.ri.cmu.edu/~fsr/FINAL_PAPERS/27_Prassler.pdf

Prassler. E. and et al. (2001) A robotic wheelchair for crowded public environments. *IEEE Robotics & Automation Magazine* 8(1): 38-45.

Röfer, T. (1997) Routemark-based navigation of a wheelchair. *Proceedings of the Third ECPD International Conference on Advanced Robotics, Intelligent Automation and Active Systems*, Bremen, Germany, 333-8.

Saffiotti, A. (1997) The uses of fuzzy logic in autonomous robot navigation. *Soft Computing* 1(4): 180-97.

Schilling, K. and et al. (1998) Sensors to improve the safety for wheelchair users. In Porrero, I. P. and Ballabio, E. eds., *Improving the quality of life for the European citizen. Technology for inclusive design and equality*, Assistive Technology Research Series 4, 331-5. Amsterdam: IOS Press.

Shanahan, J. and et al. (1999) Road recognition using fuzzy classifiers. *Proceedings of the 10th British Machine Vision Conference*, Nottingham, UK, 432-42.

Sierra, C., L´opez de M`antaras, R. and Busquets, D. (2001) Multiagent bidding mechanisms for robot qualitative navigation. *Proceedings of the Seventh International Workshop on Agent Theories, Architectures, and Languages (ATAL-2000)*, Boston, MA, 198-212.

Simon, P. and et al. (1999) The NavChair assistive wheelchair navigation system. *IEEE Transactions on Rehabilitation Engineering* 7(4): 443-51.

Soler, J. and et al. (2000) Applying the ARTIS agent architecture to mobile robot control. In Monard, M. C. and Sichman, J. S., eds., *Advances in Artificial Intelligence*, LNAI-1952, 359-68. Berlin: Springer-Verlag.

Stratmann, I. (2002) Omnidirectional imaging and optical flow. *Proceedings of the IEEE Workshop on Omnidirectional Vision*, Copenhagen, Denmark, 104-14.

Trahanias, P.E. and et al. (1997) Navigational support for robotic wheelchair platforms: an approach that combines vision and range sensors. *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, Albuquerque, NM, 1265-70.

Tunstel, E. and Jamshidi, M. (1994) Embedded fuzzy logic-based wall-following behavior for mobile robot navigation. *Proceedings of the First International Joint Conference of the North American Fuzzy Information Processing Society Biannual Conference*, San Antonio, TX, 329-30.

Ushimi, N. and et al. (2002) Online navigation of mobile robot among moving obstacles using ultrasonic sensors. In Birk, A., Coradeschi, S., and Tadokoro, S., eds., *RoboCup 2001: robot soccer world cup V* ,LNAI-2377, 477-83. Berlin: Springer-Verlag.

Whelan, P. and Molloy, D. (2000) *Machine vision algorithms in Java: techniques and implementation*. London: Springer-Verlag.

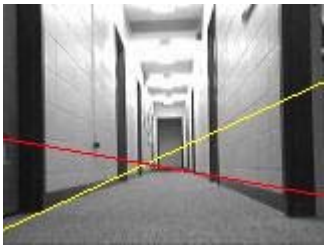
Wijesoma, W. S., Khaw, P. P., and Teoh, E. K. (2001) Sensor modeling and fusion for fuzzy navigation of an AGV. *International Journal of Robotics and Automation* 16(1): 14-25.

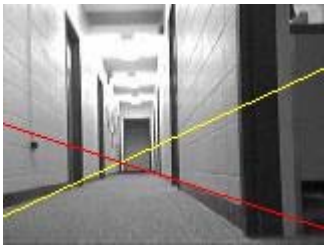
Yanco, H. A. and et al. (1995) Initial report on Wheelesley: a robotic wheelchair system. *Proceedings of the Workshop on Developing AI Applications for the Disabled*, held at the International Joint Conference on Artificial Intelligence, Montreal, Canada. http://www.cs.uml.edu/~holly/papers/ijcai95.pdf

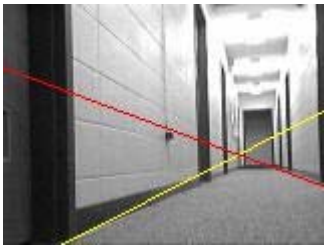
Yanco, H. A. (1998) Integrating robotic research: a survey of robotic wheelchair development. *AAAI Spring Symposium on Integrating Robotic Research*, Stanford, California.
http://www.cs.uml.edu/~holly/papers/sss98.pdf

APPENDIX A. Images Acquired by *Corridor Recognizer*

The sequence of images goes from left to right, from top to bottom. The navigation status at the bottom of each image is the decision that the corridor recognition agent made.


Corridor:Y Wall:N Object:N


Corridor:Y Wall:N Object:N

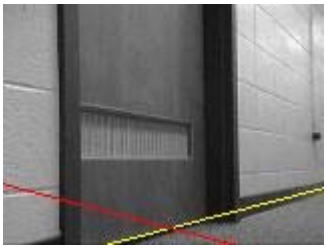

Corridor:Y Wall:N Object:N


Corridor:Y Wall:Y Object:N


Corridor:Y Wall:N Object:N


Corridor:Y Wall:N Object:N


Corridor:N Wall:N Object:N


Corridor:Y Wall:N Object:N

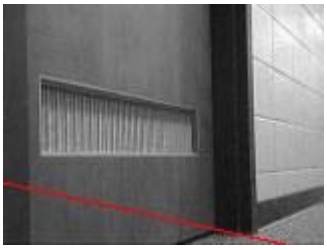

Corridor:Y Wall:N Object:N


Corridor:Y Wall:N Object:N

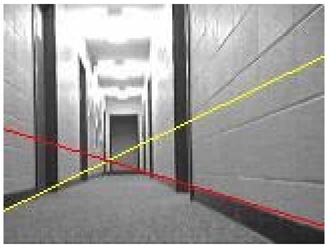

Corridor:Y Wall:N Object:N


Corridor:Y Wall:N Object:N

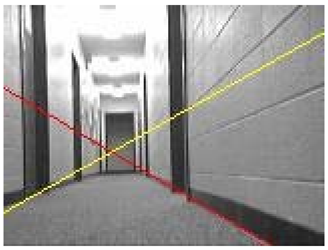

Corridor:Y Wall:N Object:N


Corridor:Y Wall:Y Object:Y

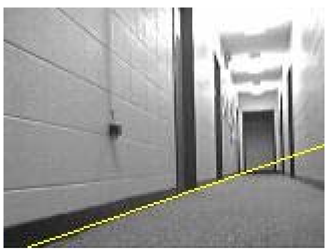

Corridor:Y Wall:N Object:N



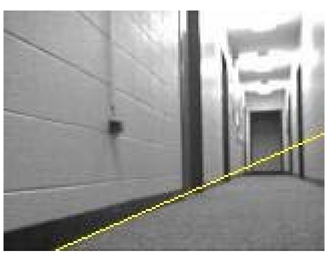
Corridor:Y Wall:N Object:N



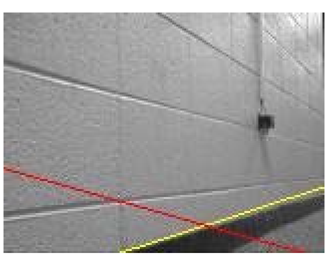
Corridor:Y Wall:N Object:N



Corridor:Y Wall:N Object:N

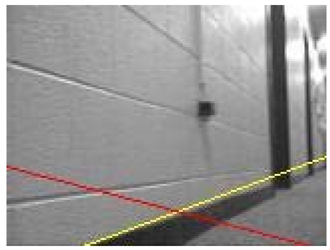


Corridor:Y Wall:N Object:N

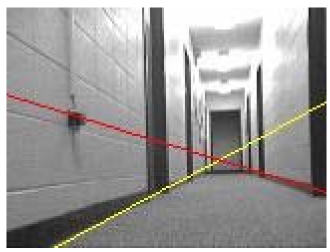


Corridor:N Wall:Y Object:N

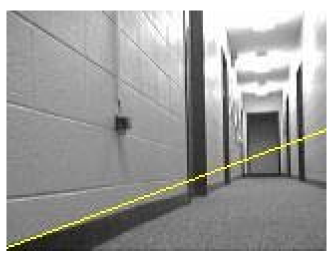
Corridor:N Wall:Y Object:N



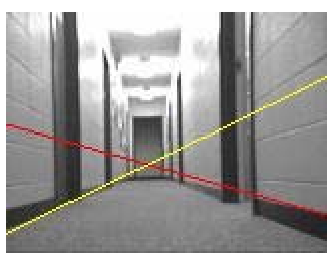
Corridor:Y Wall:N Object:N



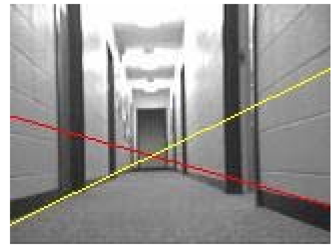
Corridor:Y Wall:N Object:N



Corridor:Y Wall:N Object:N

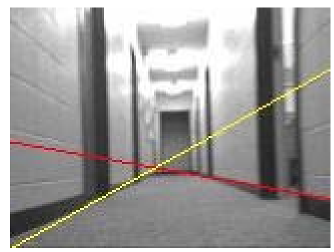


Corridor:Y Wall:N Object:N

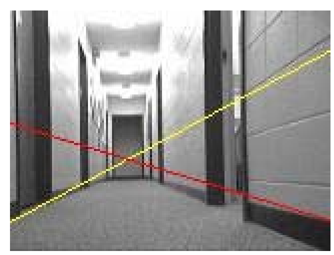


Corridor:Y Wall:N Object:N

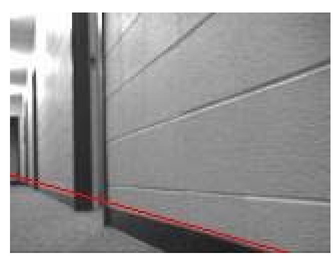
Corridor:Y Wall:N Object:N



Corridor:Y Wall:N Object:N

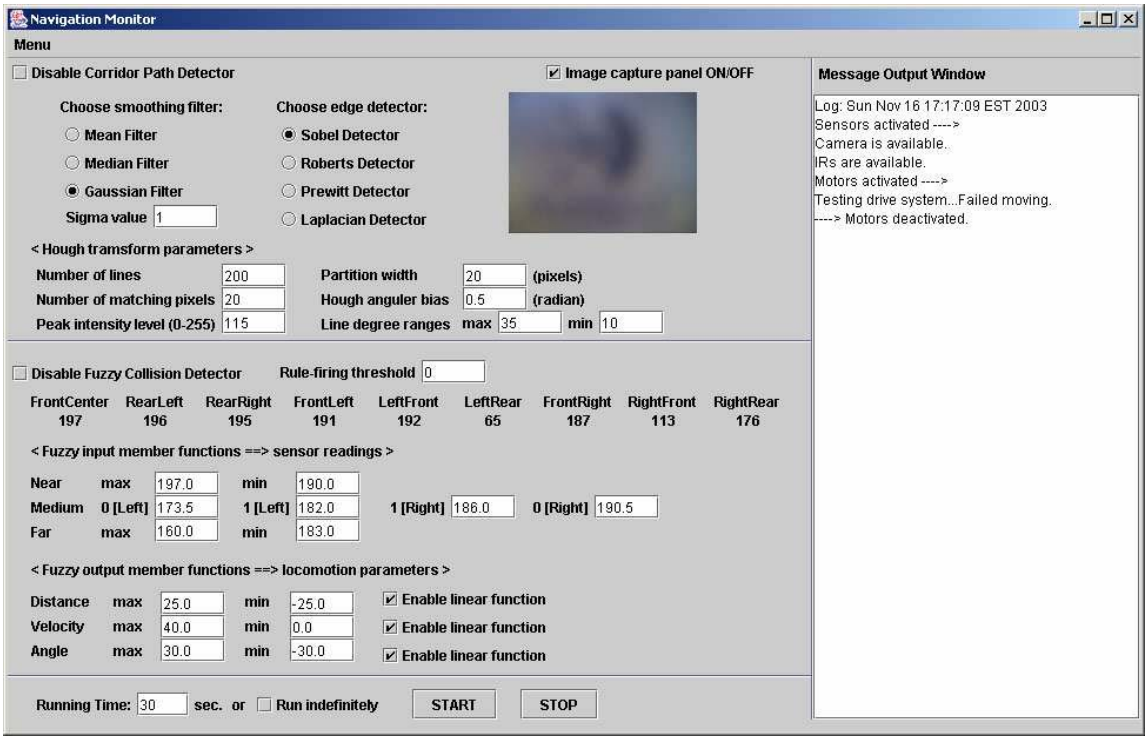


Corridor:Y Wall:N Object:N

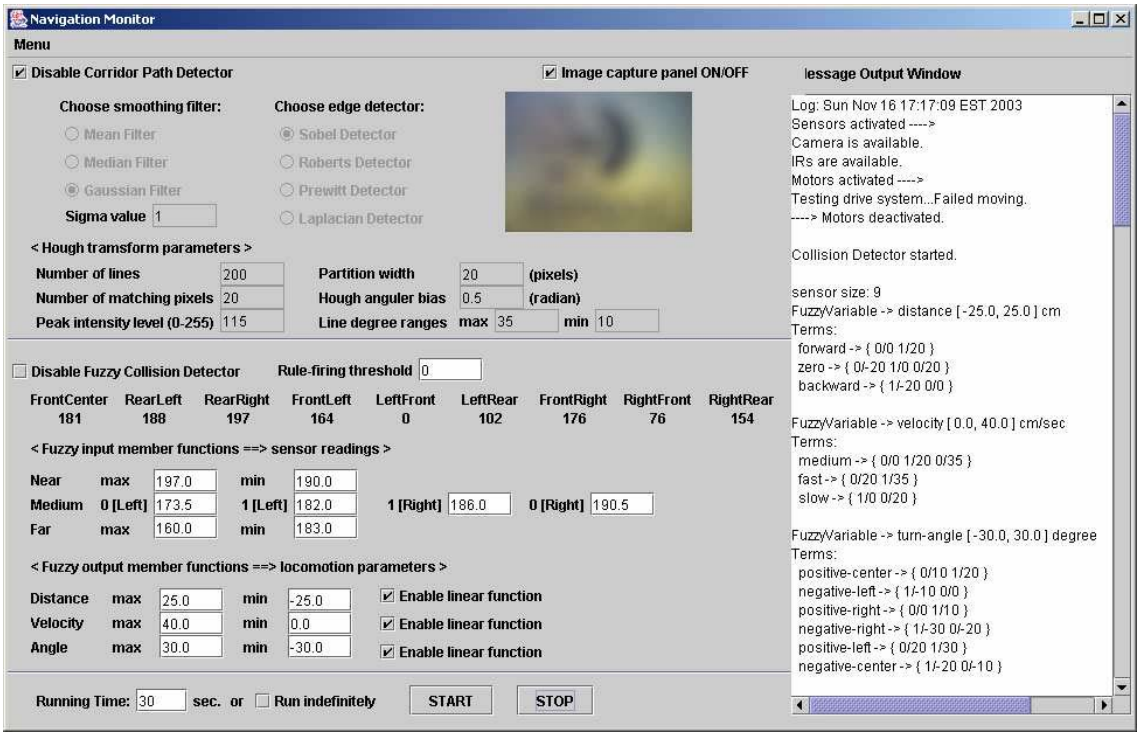


Corridor:Y Wall:Y Object:N

APPENDIX B. Control Program Interface



The GUI written in Java enables real-time perception and lets us adjust parameters for navigation. Various switches also enable agents and the features turned On/Off.



An example execution disabling Corridor Recognition Agent. Navigational information is shown on the output widow (right).

APPENDIX C. Documentation of Image Processing API

Class *ImageProcessing* contains various tools for low-level (data-driven) image processing. It is one of the APIs (Package *ugaai.wheelchair.java*) developed for the control program used in our project.

Package ugaai.wheelchair.java

Class Summary	
ImageProcessing	ImageProcessing Class provides tools for data-driven image processing.

---

ugaai.wheelchair.java

Class ImageProcessing

java.lang.Object
  |
  +--**ugaai.wheelchair.java.ImageProcessing**

---

public class **ImageProcessing**
extends java.lang.Object

ImageProcessing Class provides tools for data-driven image processing. The methods read images of any size and in any color model but only write a JPEG image in either ARGB or grayscale color model.

Field Summary	
protected double	**G_SIGMA** Sigma value for Gaussian function 1.0 as default
static int	**GRAY_BLACK** Integer representation of BLACK in grayscale
static int	**GRAY_WHITE** Integer representation of WHITE in grayscale

Constructor Summary

ImageProcessing() A contractor with the default input file "in.jpg"
ImageProcessing(java.lang.String filename) A contractor

Method Summary

void	**array2image**() Converts the image array to the image buffer in grayscale
void	**array2image**(int color) Converts the image array to the image buffer
void	**doubleThreshold**(int tmin, int tmax) Double thresholding for grayscale images
void	**doubleThresholdRGB**(int tmin, int tmax) Double thresholding for RGB images
void	**edgeDetector**(int type) Selects one of the four edge detectors and apply to images
void	**gaussianFilter**() Gaussian smoothing filter with the default sigma value 1.0
void	**gaussianFilter**(double s) Gaussian smoothing filter
int	**getHeight**() Returns the height of the input image
java.awt.image.BufferedImage	**getImageBuffer**() Returns an array of the image buffer of the input image
int[][]	**getImagePixels**() Returns an array of the image pixels of the input image
static int	**getLuminance**(int rgb) Converts a pixel value from RGB to gray-level
int	**getWidth**() Returns the width of an input image
static int[]	**histogram**(int[][] pixels, int sx, int sy, int ex, int ey) Returns an array which represents the intensity distribution histogram of the input image
void	**image2array**(int color) Converts the image buffer to the 2-D array of image pixels
boolean	**load**(java.lang.String filename) Loads an image file
void	**meanFilter**() Mean smoothing filter with window size = 3

void	**meanFilter**(int size) Mean smoothing filter
void	**medianFilter**() Median smoothing filter
void	**rgb2gray**() Converts the RGB image to the grayscale
void	**save**(java.lang.String filename) Saves the current image pixels as a JPEG file in grayscale
void	**save**(java.lang.String filename, int color) Saves the current image pixels as a JPEG image file
void	**setGaussianSigma**(int sigma) Assigns a value to Gaussian sigma
void	**setImagePixels**(int[][] new_pixels) Sets an array of the image pixels to the input image
void	**smootingFilter**(int type) Selects one of three smoothing filters and apply to the image pixel
void	**thinOperator**() Thinning operator (Non-maxima suppression)
void	**threshold**() Adaptive thresholding operator

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

G_SIGMA

protected double **G_SIGMA**

Sigma value for Gaussian function 1.0 as default.

GRAY_WHITE

public static final int **GRAY_WHITE**

Integer representation of WHITE in grayscale.

See Also:

Constant Field Values

GRAY_BLACK

public static final int **GRAY_BLACK**

Integer representation of BLACK in grayscale.

See Also:
Constant Field Values

Constructor Detail

ImageProcessing

public **ImageProcessing**(java.lang.String filename)

A constructor loads an input file and initializes the image pixels in ARGB Color Model.

Parameters:
filename - A string which contains the image input filename
Throws:
java.lang.Exception - if initialization failed

ImageProcessing

public **ImageProcessing**()

A contractor with the default input file "in.jpg".

Method Detail

getImageBuffer

public java.awt.image.BufferedImage **getImageBuffer**()

Returns an array of the image buffer of the input image.

Returns:
A BufferedImage

---

getImagePixels

public int[][] **getImagePixels**()

Returns an array of the image pixels of the input image.

Returns:
The 2-D integer array of image pixels

---

setImagePixels

public void **setImagePixels**(int[][] new_pixels)

Sets an array of the image pixels to the input image.

Parameters:
new_pixels - The 2-D integer array of an image
Returns:
None

---

getWidth

public int **getWidth**()

Returns the width of an input image.

Returns:
The image width

---

getHeight

public int **getHeight**()

Returns the height of the input image.

Returns:
The image height

---

setGaussianSigma

public void **setGaussianSigma**(int sigma)

Assigns a value to Gaussian sigma.

Parameters:
sigma - A parameter to adjust the strength of blurring
Returns:
None
See Also:
G_SIGMA - A parameter for the Gaussian smoothing filter

---

load

public boolean **load**(java.lang.String filename)

Loads an image file.

Parameters:
filename - A string which contains the image input filename
Returns:
True if the image is successfully loaded
Throws:
java.io.IOException - If the image cannot be loaded
See Also:
ImageIO.read(java.io.File)

---

array2image

public void **array2image**(int color)

Converts the image array to the image buffer.

Parameters:

color - Color Model (1 = ARGB or grayscale otherwise)

Returns:

None

---

array2image

public void **array2image**()

Converts the image array to the image buffer in grayscale.

Returns:

None

---

image2array

public void **image2array**(int color)

Converts the image buffer to the 2-D array of image pixels.

Parameters:

color - Color Model (1 = ARGB or grayscale otherwise)

Returns:

None

---

rgb2gray

public void **rgb2gray**()

Converts the RGB image to the grayscale.
To see the result, save the image as a JPEG file after applying this operator.

Returns:

None

---

save

public void **save**(java.lang.String filename, int color)

Saves the current image pixels as a JPEG image file.

Parameters:
filename - A string which contains the output file name
color - Color Model type (1 = ARGB or Grayscale otherwise)
Returns:
None
Throws:
java.io.IOException - If the image cannot be loaded

---

save

public void **save**(java.lang.String filename)

Saves the current image pixels as a JPEG file in grayscale.

Parameters:
filename - A string which contains the output file name
Returns:
None

---

histogram

public static int[] **histogram**(int[][] pixels ,int sx ,int sy ,int ex ,int ey)

Returns an array which represents the intensity distribution histogram of the input image.

Parameters:
pixels - A 2-D image pixel array
sx - x-coordinate of the staring point, x1
sy - y-coordinate of the staring point, y1
ex - x-coordinate of the ending point, x2
ey - y-coordinate of the ending point, y2

The histogram is computed for any rectangle from a starting point (x1,y1) to an ending point (x2,y2).
Returns:
The histogram of a 1-D integer array whose index represents each of the 256 gray-level

---

getLuminance

public static int **getLuminance**(int rgb)

Converts a pixel value from RGB to gray-level.

Parameters:
rgb - The RGB intensity value of a pixel
Returns:
An integer which represents the gray-level intensity of a pixel

---

threshold

public void **threshold**()

Adaptive thresholding operator automatically finds a threshold by statistically examining the intensity values of image pixels.
To see the result, save the image as a JPEG file after applying this operator.

Returns:
None

---

doubleThreshold

public void **doubleThreshold**(int tmin, int tmax)

Double thresholding for grayscale images.
To see the result, save the image as a JPEG file after applying this operator.

Parameters:
tmin - The threshold lower bound
tmax - The threshold upper bound
Returns:
None

---

doubleThresholdRGB

public void **doubleThresholdRGB**(int tmin, int tmax)

Double thresholding for RGB images.
To see the result, save the image as a JPEG file after applying this operator.

Parameters:
tmin - The threshold lower bound
tmax - The threshold upper bound
Returns:
None

---

smootingFilter

public void **smootingFilter**(int type)

Selects one of three smoothing filters and apply to the image pixel.
To see the result, save the image file as a JPEG after applying this operator.

Parameters:
type - A choice of smoothing filters
1 = Mean filter
2 = Median filter
3 = Gaussian filter

-- The default window size for Mean filter is 3x3.
-- The default Gaussian sigma is 1.0.
Returns:
None

---

meanFilter

public void **meanFilter**(int size)

Mean smoothing filter computes the local mean (averaged) value in the given window.
To see the result, save the image as a JPEG file after applying this operator.

Parameters:
size - The size of a window
Returns:
None

---

meanFilter

public void **meanFilter**()

Mean smoothing filter with window size = 3
To see the result, save the image file as a JPEG file after applying this operator.

Returns:
None

---

medianFilter

public void **medianFilter**()

Median smoothing filter.
To see the result, save the image as a JPEG file after applying this operator.

Returns:
None

---

gaussianFilter

public void **gaussianFilter**(double s)

Gaussian smoothing filter.
To see the result, save the image as a JPEG file after applying this operator.

Parameters:
s - The value for Gaussian sigma
Returns:
None
See Also:
G_SIGMA

---

gaussianFilter

public void **gaussianFilter**()

Gaussian smoothing filter with the default sigma value 1.0

To see the result, save the image as a JPEG file after applying this operator.

Returns:
None

---

edgeDetector

public void **edgeDetector**(int type)

Selects one of the four edge detectors and apply to images.
Modifying detector masks in the code may or may not enhance the result.
To see the result, save the image as a JPEG file after applying this operator.

Parameters:
type - A choice of edge detectors
1 = Sobel edge detector
2 = Roberts edge detector
3 = Prewitt edge detector
4 = Laplacian edge detector

Returns:
None

---

thinOperator

public void **thinOperator**()

Thinning operator (Non-maxima suppression) is applied after performing edge detection.
To see the result, save the image as a JPEG file after applying this operator.

Returns:
None
