Beobot 2.0: Cluster Architecture for Mobile Robotics

Christian Siagian, Chin-Kai Chang, Randolph Voorhies, and Laurent Itti
Department of Computer Science, University of Southern California, Los Angeles, California 90089
e-mail: siagian@usc.edu, chinkaic@usc.edu, voorhies@usc.edu, itti@pollux.usc.edu

Received 6 April 2010; accepted 23 October 2010

With the recent proliferation of robust but computationally demanding robotic algorithms, there is now a need for a mobile robot platform equipped with powerful computing facilities. In this paper, we present the design and implementation of Beobot 2.0, an affordable research-level mobile robot equipped with a cluster of 16 2.2-GHz processing cores. Beobot 2.0 uses compact Computer on Module (COM) processors with modest power requirements, thus accommodating various robot design constraints while still satisfying the requirement for computationally intensive algorithms. We discuss issues involved in utilizing multiple COM Express modules on a mobile platform, such as interprocessor communication, power consumption, cooling, and protection from shocks, vibrations, and other environmental hazards such as dust and moisture. We have applied Beobot 2.0 to the following computationally demanding tasks: laser-based robot navigation, scale-invariant feature transform (SIFT) object recognition, finding objects in a cluttered scene using visual saliency, and vision-based localization, wherein the robot has to identify landmarks from a large database of images in a timely manner. For the last task, we tested the localization system in three large-scale outdoor environments, which provide 3,583, 6,006, and 8,823 test frames, respectively. The localization errors for the three environments were 1.26, 2.38, and 4.08 m, respectively. The per-frame processing times were 421.45, 794.31, and 884.74 ms respectively, representing speedup factors of 2.80, 3.00, and 3.58 when compared to a single dual-core computer performing localization.

1. INTRODUCTION

In the past decade, researchers in the field of mobile robotics have increasingly embraced probabilistic approaches to solving hard problems such as localization (Fox, Burgard, Dellaert, & Thrun, 1999; Thrun, Fox, & Burgard, 1998; Thrun, Fox, Burgard, & Dellaert, 2000), vision (Heitz, Gould, Saxena, & Koller, 2008; Wu & Nevatia, 2007), and multirobot cooperation (Fox, Burgard, Kruppa, & Thrun, 2000; Thrun & Liu, 2003). These algorithms are far more sophisticated and robust than the previous generation of techniques (Brooks, 1986; Maes & Brooks, 1990; Pomerleau, 1993). This is because these contemporary techniques can simultaneously consider many hypotheses in forms of multimodal distributions. Because of that, however, they are also far more computationally demanding. For example, a visual recognition task, in which we need to compare the current input image captured by a camera against a large database of sample images (Bay, Tuytelaars, & Gool, 2006; Lowe, 2004; Mikolajczyk & Schmid, 2005), requires not only that robust visual features be extracted from the input image—which already is a computationally demanding task—but also that these features be matched against those stored in the database—an even more demanding task when the database is large. As a point of reference, comparing two 320×240 images using scale-invariant feature transform (SIFT) features (Lowe, 2004) can take 1–2 s on a typical 3-GHz single-core machine. To be able to run such algorithms in near real time, we need a mobile robot equipped with a computing platform significantly more powerful than a standard laptop or desktop computer.

However, existing indoor and/or outdoor mobile robot platforms commercially available to the general research community still appear to put little emphasis on computational power. In fact, many robots, such as the Segway RMP series (Segway, Inc., 2009), have to be separately furnished with a computer. On the other hand, robots that come equipped with multiple onboard computers either do not use the most powerful computers available today [e.g., the Seekur (MobileRobots, Inc., 2009), which relies on the less powerful PC/104 standard] or have fewer computers (e.g., Carnegie Mellon University Robotics Institute, 2009; Willow Garage, 2009) than our proposed solution.

Before describing the design and implementation of our robot, in Section 1.1 we survey the current trends in the mobile robot market and identify the most desirable features in an ideal research robot (aside from our central requirement of powerful computational facilities). Note that some of the robots discussed below may no longer be available (or may never have been) to the general public. We include them nonetheless for completeness of our analysis.

We then describe our main contribution in Section 1.2, the design and implementation of our proposed platform, Beobot 2.0, a powerful mobile robot platform equipped with a cluster of 16 2.2-GHz processing cores. Our robot uses compact Computer on Module (COM) processors with modest power requirements, thus accommodating

various robot design constraints while still satisfying the requirement for computationally intensive algorithms.

Our complete design specifications, including supplier and cost information for almost all the materials, are freely available on the Internet (Siagian, Chang, Voorhies, & Itti, 2009). As the manufacturing, assembly, and machining details are available online, in this paper we focus on (1) the design decisions we made, the implementational issues we faced, and how we resolved them, and (2) experimental testing of the robot in diverse tasks.

1.1. Current Mobile Robot Platforms

In the current state of robotics, researchers utilize a variety of mobile robots, from the commercially available (iRobot Corporation, 2009b; MobileRobots, Inc., 2009; Willow Garage, 2009) to the custom made (Carnegie Mellon University Robotics Institute, 2009). These robots are built for many different environments, such as underwater (iRobot Corporation, 2010; USC Robotics, 2009), aerial (Finio, Eum, Oland, & Wood, 2009; He, Prentice, & Roy, 2008), and land (Quigley & Ng, 2007; Salichs, Barber, Khamis, Malfaz, Gorostiza, et al., 2006). Here we focus on land robots of a size close to that of an adult human that can traverse most urban environments, both indoors and outdoors, and for considerable distances. In addition, it is versatile enough for research in many subfields such as localization/navigation, human–robot interaction, and multirobot cooperation.

Furthermore, we primarily focus on sites that are similar to a college campus setting, which is mostly paved with some rough/uneven roads, not terrains that one would see in combat zones. Nowadays, because there is a concerted effort by most governments to make pertinent locations accessible to the disabled (using wheelchairs), legged robots [from the small QRIO and AIBO by Sony (Sony Entertainment Robot Europe, 2009) to the human-sized Honda Asimo (American Honda Motor Co., Inc., 2009)] are no longer a must. A wheeled platform would suffice for the target environments. However, the ability to traverse reasonably sloped terrain (about 10 deg) should also be expected. Also, some form of weather protection in the outdoors is essential. Although the robot is not expected to operate in all kinds of harsh weather (pouring rain, for example), like Seekur and Seekur jr. by MobileRobots (MobileRobots, Inc., 2009) and IRobot's PackBot (iRobot Corporation, 2009a), it should nevertheless be able to handle most reasonable conditions.

An overall size that is close to that of an adult human is ideal because the robot would be small enough to go through narrow building corridors and yet large enough to travel between buildings in a timely manner. And thus we exclude small robots such as the Khepara (AAI Canada, Inc., 2009) or large robotized cars such as the entries to the DARPA Grand Challenge. Smaller robots such as the Roomba (iRobot Corporation, 2009b) and Rovio (Evolution Robotics, Inc., 2009) and slightly larger ones such as the Pioneer (MobileRobots Inc., 2009) are also excluded because of their lower payload capacity, which limits the amount of computing that can be carried to a single laptop.

Aside from mobility, a few other important features contribute to the usability of the robot. They are battery life, sensors, interfaces, and available software. An ideal battery system would be one that enables the user to run for a whole day without having to recharge. The two factors that matter here are the total charge carried and the amount of charge required to operate the robot. The latter is dictated by the total weight of the robot and power consumption of the computers and devices. These requirements should be decided first. On that basis, the former can then be adjusted by selecting the proper battery system (type and quantity).

There are different types of available batteries: NiCd, NiMH, sealed lead acid (SLA), and lithium based. The trade-off is that of cost, dimensions, and durability. For one, SLA batteries are the most economical (in terms of cost-to-charge ratio), widely available when it comes time to replace them, and robust as they are easy to maintain and long lasting. However, SLA batteries have low charge-to-weight as well as charge-to-volume ratios compared to, particularly, the lithium-based technologies. Lithium batteries are lighter and more compact for a comparable amount of charge (National Institute of Standards and Technology, 2009). However, these types of batteries are much more expensive and fragile than the very rugged SLAs. If lithium batteries are not handled carefully, for example by not using protective circuits, they can explode. Although for the size of robot we are considering battery weight is not as much an issue, note that volume would still matter in terms of placement.

It is important to have easy access to the battery compartment so that we do not have to unscrew or disassemble components in order to charge the batteries. If the batteries can be charged rapidly, within an hour or two, an even better option would be to be able to do so without having to remove them from the robot, instead using a docking station or a wall plug-in outlet. If rapid recharge is not available, a feature to hot swap the batteries as in Willow Garage (2009) to avoid shutting down computers in the switching process would be convenient.

For a platform to be applicable for a wide range of robotics and vision research, most commercial robots are furnished with a variety of sensors and manipulation tools such as a robot arm and also provide avenues for future expansion. When selecting a sensor, we look for compact, light, low-power devices that exhibit high accuracy and high update rates. Popular sensors such as laser range finders, sonar rings, cameras, inertial measurement units (IMU), global positioning systems (GPS), and compasses should be considered as potential accessories. For a camera in particular, negligible latency is a must. After a number of experiments, we found that Firewire (IEEE-1394) cameras were the best in minimizing delays, more so than Internet

protocol (IP) or universal serial bus (USB) cameras. In addition, related features such as pan–tilt–zoom, autofocus, image stabilization, low-light capability, and wide-angle or omnidirectional viewing setup are also made available by various companies.

As for sensor expansion, aside from anticipating the future extra payload, it is also important to have many accessible USB ports placed throughout the body of the robot and USB-to-serial converters for serial devices, as well as several microcontrollers that can preprocess slower input signals.

Another important feature is having multiple types of user interface. For example, USB inputs are useful to connect a keyboard and monitor to the computers in the robot to allow for hardware and operating system reconfiguration. In addition, many robots have reasonably sized (15–25 cm) full red–green–blue (RGB) liquid crystal display (LCD) monitors for visualization of the robot's state during test runs. Furthermore, wireless network connections for remote secure shell (SSH) logins give us additional flexibility to allow for safe and faster algorithm on-site debugging. At the same time, the robot can use the external connection to access outside network or Internet resources, which can be useful in some scenarios. A related feature in this category is a standard radio frequency (RF) remote controller for stopping the robot whenever autonomous driving starts to fail. Furthermore, most robots (Carnegie Mellon University Robotics Institute, 2009; MobileRobots, Inc., 2009) are equipped with large kill switches to stop the flow of power to the motors.

In addition to the hardware-related aspects, robotic companies also provide software libraries to conveniently access all the included devices and monitor low-level states such as battery charge and temperature. Some companies (MobileRobots, Inc., 2009) provide further value additions such as mapping and navigation tools and even a full-blown simulation environment. We list the common software offerings in Section 4, where we describe our freely available toolkit (Itti, 2009).

1.2. Our Approach

Beobot 2.0 is the next iteration of the Beobot system developed in our lab (Chung, Hirata, Mundhenk, Ng, Peters, et al., 2002). The original Beobot integrated two full-sized, dual-CPU motherboards for a total of four 1-GHz processors. For Beobot 2.0, we use eight dual-core COM systems. Each COM measures just 125 × 9.5 × 18 mm and nominally consumes only 24 W of power. Nonetheless, with a 2.2-GHz dual-core processor, a COM has the computing power equivalent to current dual-core laptop systems. Despite this state-of-the-art computing platform, we have managed to keep the overall cost of our research-level, cluster-based mobile robot to under $25,000 (detailed in Siagian et al., 2009).

One aspect of a COM system to underscore here is the ease with which its components can be upgraded. Because the input and output signals are routed through just two high-density connectors, one need only remove the current module and replace it with an upgraded one. Thus, as more and more powerful processors become available, Beobot 2.0's computer systems can keep pace, making it somewhat more resistant to the rapid obsolescence that is characteristic of computer systems. The ability to keep pace with processor technology is important because robotic algorithms are expected to continue to evolve and become ever more complex, thus requiring commensurate levels of computing power.

Beobot 2.0's computer system is mounted on an electric wheelchair base (Figure 1), with an overall size that is close to that of a human. This allows the robot to navigate through corridors and sidewalks and creates an embodiment that is ideal for interacting with people. We assume that the majority of these pertinent locations are wheelchair accessible, as required by law. We believe that even with this locomotion limitation, there are still enough physically reachable locations to perform comprehensive real-world experiments. Figure 1 shows the finished robot.

The rest of the paper is organized as follows: first, we describe the electrical system in Section 2 and then the mechanical system in Section 3. Section 4 goes into the details of our software library, highlighting the advantage of implementing a computing cluster in robotics research.

In Section 5 we examine the robot on various important operational aspects, the most important of which is computational speed/throughput, to demonstrate how one could benefit from such a complex computing cluster architecture. We test Beobot 2.0 using three benchmark algorithms. One is the popular SIFT (Lowe, 2004) object recognition. The second is a distributed saliency algorithm (Itti, Koch, & Niebur, 1998), which models the visual attention system of primates. The algorithm operates on a very large image of 4,000 × 4,000 pixels and returns the most salient parts of the image. The last one is a vision localization system by Siagian and Itti (2009) that requires the system to compare a detected salient landmark input with a large landmark database obtained from previous visits. All of these algorithms are part of the Vision Toolkit, available freely online (Itti, 2009), which also houses Beobot 2.0's software control architecture, including obstacle avoidance (Minguez & Montano, 2004) and lane following (Ackerman & Itti, 2005). We then summarize our findings (in Section 6) and what we have learned through the process of building this robot.

2. ELECTRICAL SYSTEM DESIGN AND IMPLEMENTATION

Figure 2 presents an overview of the electrical system. On the right-hand side of the figure, there are two baseboards, each housing four COM Express modules (explained in

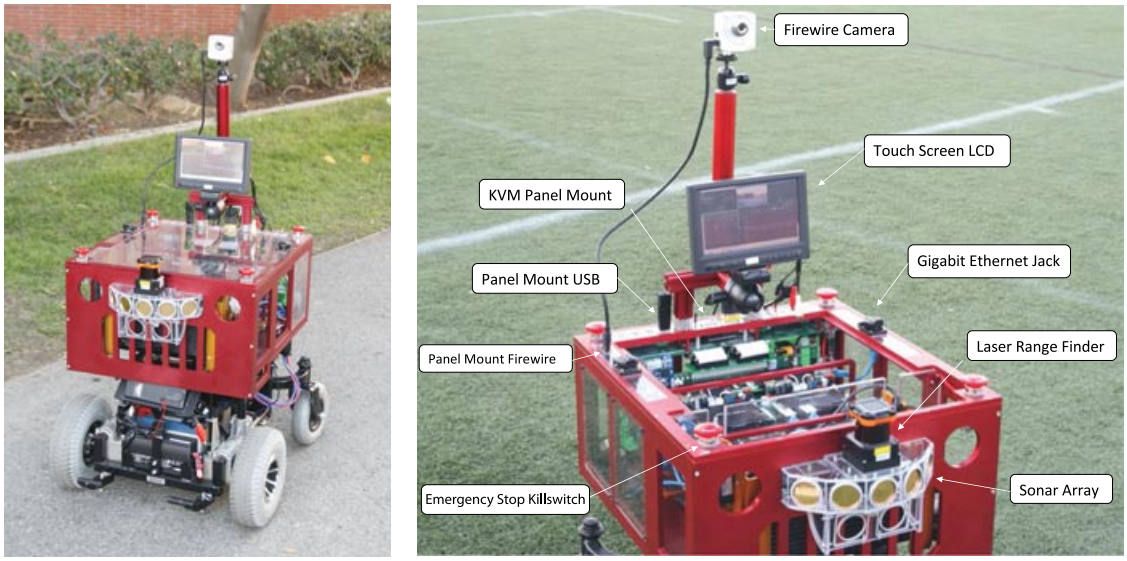


Figure 1. Various features of Beobot 2.0. Beobot 2.0 utilizes an electric wheelchair platform to carry a high-performance computing cluster of 16 processor cores, 2.2 GHz each. The robot is equipped with various sensors such as Firewire camera, laser range finder, sonar array, IMU, compass, and GPS. In addition, panel-mount waterproof USB connectors are available for new sensors, along with RJ45 Ethernet for wired Internet connection and panel-mount KVM inputs for regular-sized monitor, keyboard, and mouse. There is also a touchscreen LCD for a convenient user interface. Furthermore, the kill switches at each corner of the robot are available as a last resort to stop it in emergency situations.

depth below), and implementing signals such as gigabit Ethernet for the backplane intermodule communication, as well as others such as SATA (two per module), PCI Express, USB, and VGA. Beobot 2.0 uses a PCI Express 1394-Firewire card for a low-latency camera connection. One of the SATA ports was used for the primary hard drive and the other for external drives such as CD-ROM (useful for installing operating systems, for example). Giving each module its own hard drive obviates the need to pass around copies of stored data, such as large knowledge databases obtained during training.

There are six USB signal implementations per computer for a total of 48. Some of them are being used for sensors listed in Table I. Several of the USB connectors are panel mounted outside the robot for ease of connecting external devices using dust- and waterproof connectors (Figure 1). In addition, there are also USB connectors inside, on the baseboards (see Figure 3).

Furthermore, there is an onboard keyboard-video-mouse (KVM) switch to toggle between each of the eight computers. The KVM is an eight-to-two switch, eight computers to two display outputs. The display signal outputs can be either a regular-sized external monitor or to an onboard 8-in. touchscreen LCD with a full-color video graphic array (VGA) interface (Figure 1). Note that in practice we operate all computers from a single node using an SSH login session to conveniently run and monitor multiple programs simultaneously. The use of wired interface to the individual computers is usually limited to hardware, BIOS, and boot troubleshooting.

Table I. Sensors provided in Beobot 2.0.

Item	Company name	Reference
Laser range finder	Hokuyo	Hokuyo Automatic Co., Ltd., 2009
IMU	MicroStrain	MicroStrain, Inc., 2009
Compass	PNI	PNI Sensor Corporation, 2009
Sonars (7 units)	SensComp	SensComp, Inc., 2009
GPS	US Global Sat	USGlobalSat, Inc., 2009

The objectives for selecting a computing platform appropriate for the robot are high computing power, compactness, and low energy consumption. To have close to maximum achievable speed, we concentrate on the X86 architectures rather than far less powerful CPU types such as ARM or Xscale. Within the X86 family, we select the mobile processor version rather than its desktop counterpart for energy efficiency, still being competitive in computing power. By the same token, in using the mobile CPU version, the corresponding embedded systems option can be selected

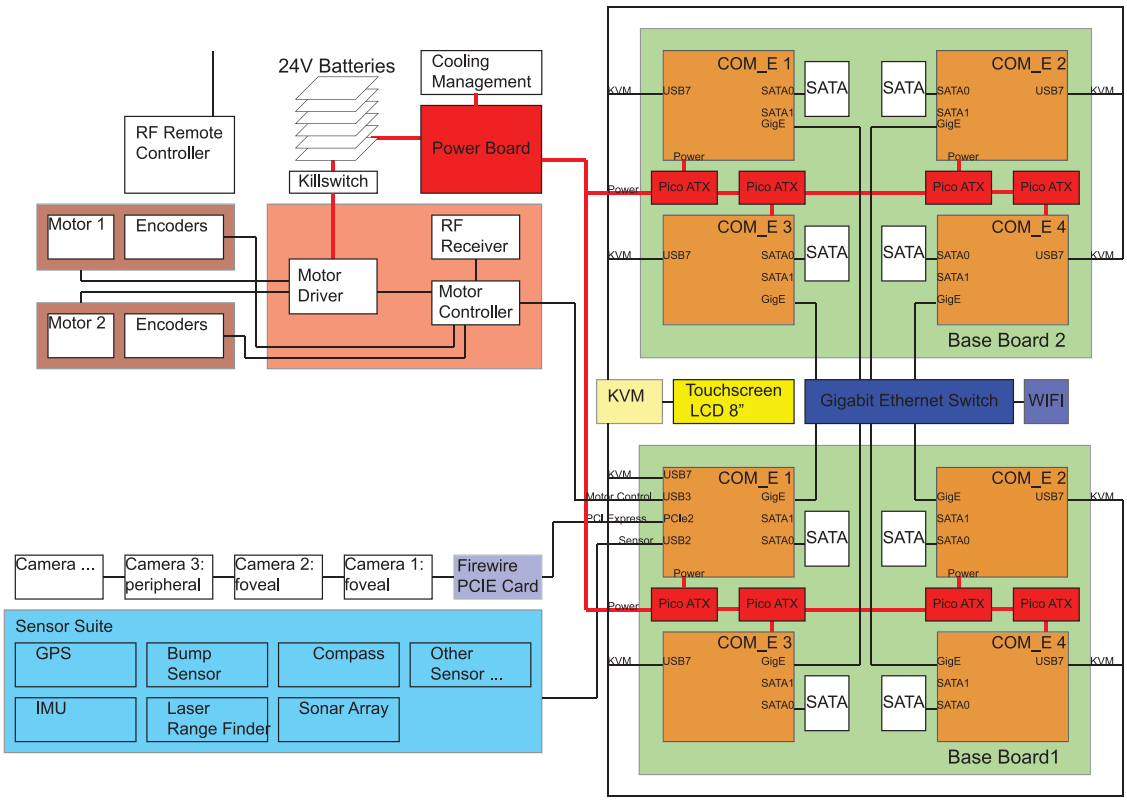


Figure 2. Beobot 2.0 electrical system. On the right-hand side of the diagram, there are two baseboards, each housing four COM express modules and each module with its own SATA hard drive. The backbone intercomputer communication is gigabit Ethernet that is connected through a switch. For visual interface to individual computers, a KVM is used to connect to either an 8-in. LCD touchscreen or an external monitor, mouse, and keyboard. In addition, a PCI Express–Firewire interface card is used to connect to a low-latency camera. The other sensors are connected via the many USB connectors that are panel mounted on top of the robot as well as on the baseboard. The whole system is powered by a 24-V battery circuit supply (with kill switches for safety purposes) and is regulated through a set of dedicated Pico-ATX power modules. The same battery circuit also powers the motors as well as the liquid-cooling system.

for the mobile platform (regular-sized motherboards do not usually use mobile CPUs), which resolves the size issue.

In the family of embedded systems, there are two types of implementations. The first family of systems have the interfaces already implemented, ready to use. An example is the ITX form-factor family (pico-ITX, nano-ITX, mini-ITX) (Via, 2009). The drawback is that the provided interfaces are fixed. They may not be the specific ones that are needed, and unused connections can be a waste of size as we cannot customize their location and orientation. In addition, by using off-the-shelf motherboards, their dimensions have to be accommodated in the design specifications, which may also limit the options for the locomotion platform.

In contrast, the second type of embedded systems, the COM concept, provides specifications for all the interfaces only through a set of high-density connectors. These specifications are usually defined by an industry consortium such as the XTX-standard (XTX Consortium, 2009). The actual breakout of the individual signals (such as gigabit Ethernet, USB, PCI Express) from the module connectors to the outside devices (a hard drive, for example) has to be done on a custom-made carrier board. By building custom baseboards, the overall size of the electronics can be controlled by implementing only those signals that we actually need. In addition, connector placement (as well as type) can be specified so as to minimize the amount of cabling in the system.

In the end, we found that a COM module solution best met our requirements, which we stated at the start of this section. Within this class, there are three options: ETX (ETX Industrial Group, 2009), XTX (XTX Consortium, 2009), or COM Express (COM Express Extension, 2009). These modules use the most powerful processors, as opposed to the smaller but less powerful systems such as PC104-based

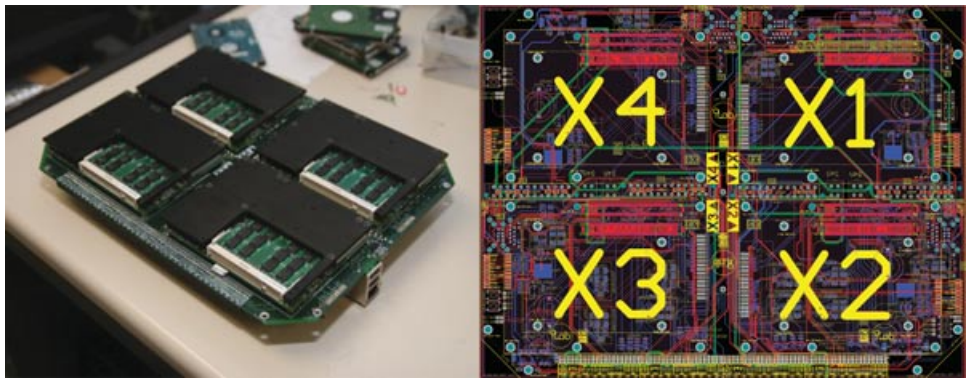


Figure 3. Baseboard. The image on the left is a fully assembled baseboard with four COM Express modules. The black plates are the heat spreaders attached to the processors. There is also an Ethernet and two USB jacks placed on the right-hand side of the board. The layout on the right is the circuit done in Altium (Altium Limited, 2009) PCB design software.

Qseven (Qseven Standard, 2009). We chose COM Express because it has an onboard gigabit Ethernet interface on the module, and it is only slightly larger (12.5-cm length × 9.5-cm width) than the XTX and ETX module (11.5-cm length × 9.5-cm width). Gigabit Ethernet is critical because in a cluster architecture, intercomputer communication can be just as important as the computing power of individual nodes. If the communication procedure cannot provide data fast enough, the individual processors will simply idle most of the time, waiting for data to arrive. This is especially true in our case because Beobot 2.0 is designed to perform heavy-duty, real-time vision computation in which the real-time video streaming is much more demanding than sending intermediate results.

We implemented two carrier/baseboards (refer to Figure 3), each accommodating four COM Express modules. A total of eight modules is chosen because the computing system fits within the mobile platform and because this number is expected to suffice for our research needs based on the findings presented in Section 5.

We used the Kontron COM Express design guide (Kontron, 2007) [for the Kontron ETX-Express-MC 2.2 GHz (T7500) COM Express module (Kontron, 2009)] to help properly design the electronic circuits as well as lay out the components in the board. We used the electronics computer-aided design (ECAD) layout software Altium (Altium Limited, 2009) to plan the physical placement of all the desired devices and connectors with as little cabling as possible for a system of eight computers. Altium's three-dimensional (3D) visualization proved to be an invaluable feature as it allowed us to verify that boards and devices packed close together in the robot would not collide or otherwise interfere with any other components.

The most critical part in successfully implementing the baseboards was being able to take care of the high-speed differential-pair signal requirements such as matching length and spacing, as well as minimizing the number of vias in the baseboard. Altium allows its users to specify rules for each trace on the board, which tremendously eases the process of identifying unsatisfied constraints. We found that the signals are quite robust as long as the stated requirements are adhered to. In addition, most of these signals need very few supporting circuits. The most components required by a signal is eight, for the USB current limiter (500 mA) circuit. The VGA signal actually specifies that it needs a filtering circuit with many components, but the KVM chip furnishes this feature.

To provide clean and fail-safe power given a supply from the available batteries, a Pico-ATX (Ituner Networks Corp., 2009) module [see Figure 4(a)] is used to regulate power to each COM Express for a total of eight. There is also one extra Pico-ATX powering all the peripheral boards

(a) Pico-ATX (b) KVM board (c) Liberty312 battery and charger

Figure 4. Various power-related components.

and sensors. There are three peripheral boards: one to control the cooling system, one to control access to the motors, and a sensor board that houses all the various built-in sensors. The power to the drive-train motors does not need to be filtered and, thus, is directly connected from the batteries. Because power supplies have a high rate of failure, going with multiple power modules provides better granularity in that if one module fails, the remaining seven computers can still run. Additionally, the lower individual supply requirement allows for a wider range of products to choose from than would have been available in a single module solution.

Table II summarizes the important electrical features: interprocessor communications, input/output interfaces, KVM interface, sensors, and power management. These features are shown to be critical while compiling the list of available commercial robots as well as from our experience conducting robotics research.

3. MECHANICAL SYSTEM DESIGN AND IMPLEMENTATION

The mechanical design of the robot is divided into two parts: the locomotion platform, which is the dark-colored robot base in Figure 5 and described in Section 3.1, and the computing cluster housing, which is the cardinal-colored structure described in Section 3.2.

Again, note that we created a wiki page (Siagian et al., 2009) to detail the execution matters such as actual part drawings (SolidWorks Corp., 2009), part manufacturing through a machine shop, or finding suppliers for the needed devices listed in the bill of material.

3.1. Locomotion System

For the locomotion platform, we selected a Liberty 312 electrical wheelchair (Major's Mobisist, 2009) instead of building one from scratch. Often priced at thousands of dollars, these types of units are easily acquired second hand through channels such as Craigslist or eBay (ours cost U.S. $200). The wheelchair is a robustly engineered, stable, safe, and low-maintenance platform. Most importantly, adhering to the wheelchair form factor allows the robot to traverse most terrain types encountered in modern urban environments, both indoors and out. This platform can also carry heavy payloads (113–136 kg), which means the ability to add many more devices to the robot's computing cluster. An important factor to consider is the ability to control the motors over a wide range of speeds (0–16.09 km/h) with good resolution in between. The wheelchair platform has this characteristic as it is designed for fine-grained control, as opposed to the remote control (RC) car used by the original Beobot (Chung et al., 2002), which could be driven only at maximum speed. Another benefit of the wheelchair is that it places the computing cluster on top, relatively high above the ground (about 50 cm) and away from the thick dust and mud that can accumulate on the street. Note that the robot's driving dynamics is taken care of because the wheelchair is designed to have a person on top, where the computing system now is placed. This is accomplished by the wide-spacing configuration of the wheels, enveloping the payload, to allow for the overall balance of the system while it is moving reasonably fast. In addition, the heavy SLA batteries are placed on the bottom to lower the center of mass.

To control the wheelchair, we designed a motor board to connect the battery and motors to inputs from the computer for autonomous control as well as to a 2.4-GHz remote controller (RC) for manual driving or overriding. A dual-output motor-driver named Sabertooth (Dimension Engineering LLC, 2009) is used to provide up to 25 A to each motor. In addition, because the motor driver has a built-in electrical brake system, the mechanical brakes that stop the motors by pinching the back shafts are taken off. This then allows the back shafts to be coupled to a pair of encoders to provide odometry data. As a safety precaution, Beobot 2.0 is furnished with four kill switches (Figure 1), one on each corner for the user to immediately stop the robot in the event of an emergency.

The wheelchair comes with a pair of 12-V, 35-Ah SLA batteries, connected in series to provide a 24-V supply. They have a form-factor space of 19.5-cm length $\times$ 13.2-cm width $\times$ 15.5-cm height for each battery. An attractive feature of the wheelchair is the built-in, wall-outlet, easy-plug-in battery recharging system, shown in Figure 4(c). With this, the batteries can be conveniently recharged without having to put them in and take them out of the robot, although the recharging process does take an average of 10 h.

3.2. Computing Cluster Case

The structure surrounding the computing clusters, as shown in Figure 5, shields the computing cluster from unwanted environmental interference such as dust and mud. The structure is divided into two isolated chambers as illustrated in the figure. The back chamber is the watertight area where the cluster is placed. The front chamber is an open area, reserved for a liquid-cooling system (further elaborated in Section 3.2.2), which includes a radiator to allow for maximum air flow. These two cooling subsystems are connected through Tygon tubing for liquid flow and are physically held together by a pair of aluminum holders. The computing cluster, along with the cooling system, itself is mounted on shock-absorbing standoffs (Section 3.2.1) to withstand violent collision in the rare event the robot hits an obstacle.

3.2.1. Vibration Attenuation and Shock Absorption System

As illustrated in Figure 5, the only connections between the computing system and the robot base are the

Table II. Beobot 2.0 electrical system features.

Feature	Our requirements	Solution chosen	Alternative considered	Positives of chosen solution	Negatives of chosen solution	Remarks
Intermodule communication	Large enough bandwidth to stream real-time video	Gigabit Ethernet	Symmetric multiprocessing (SMP) architecture, same memory module, shared throughout the system bus	Simpler to build	Larger latency	Gigabit Ethernet network also connected to wireless Internet connection for remote logins
Input/output interfaces	Connection to gigabit Ethernet, PCI Express, SATA, USB, and VGA signals from the COM Express modules	Custom mother/baseboards and peripheral boards connected to easily accessible panel-mount dust and waterproof connectors	Cables for access from COM Express modules to the connectors	Minimal cabling allow for easier debugging/repair	Complex design, especially in implementing high-speed signals with differential pair requirements	Found that the signals are quite robust as long as the stated requirements [in the design guide (Kontron, 2007)] are adhered to
KVM computer interfaces	Provide easy visualization and access to each computer	Integrated KVM system	Individual VGA, mouse, keyboard, cables	Do not need to cram cables to eight computers into a small area	Complex design, integrated in the baseboard and peripheral boards	8 computers to 2 display KVM switches: one for regular-sized external monitor (if desired), another onboard, 7 in full-color VGA touchscreen LCD; note that software remote access solution (using SSH, for example) is also available

(*Continued*)

Table II. Continued

Feature	Our requirements	Solution chosen	Alternative considered	Positives of chosen solution	Negatives of chosen solution	Remarks
Sensors	Need sensors for a wide range of robotics and vision research	Integrate sensors to the design	Add sensors later	Less likely to accommodate space for sensors	Upfront added cost weight and design complexity	We include versatile sensors that are widely used; also has an abundance of accessible USB ports throughout robot body and microcontrollers for new sensors
Power Management	Clean and fail-safe power (12, 5, and 3.3 V) as per COM Express Design Guide (Kontron, 2007); these modules primarily need large supply of 12 V, up to 5-A maximum per module, making for a total of 40 A (480 W) for all eight modules	Off-the-shelf system: Pico-ATX (Ituner Networks Corp, 2009) modules to power each COM Express board (plus 1 for all the peripherals and sensors)	Built custom power system and profile supplying high current on the voltages needed the most	Fully engineered and tested, with protection from overdischarging and overpeaking; shorter prototyping cycles; end up with about the same cost	Need to accommodate shape and sizes, waste of energy for unneeded supplies (−5 and −12-V CC), and lack of availability of a 12-V supply that goes over 12 A	Modified the compact Pico-ATX and routed the power traces appropriately in the baseboard to minimize cable length; they fit in the available space without requiring any alterations to the mechanical design

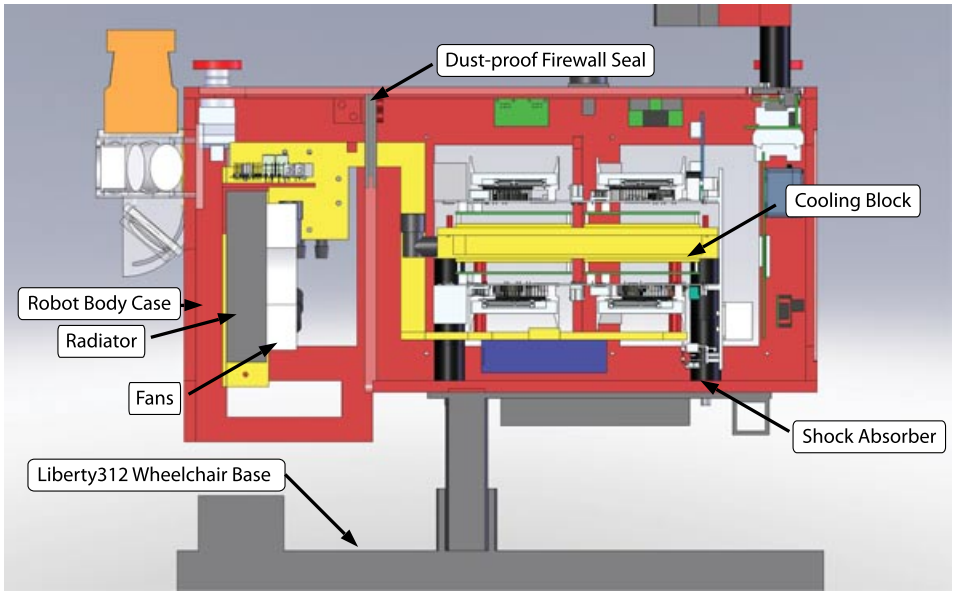


Figure 5. A SolidWorks (SolidWorks Corp., 2009) model of the robot shown from its side and cut through its center. The bottom of the image displays the Liberty312 wheelchair base, and above it is the robot body in cardinal color. The robot body is divided into two chambers by the dust-proof firewall. The back chamber completely seals the computers within from the elements. The front of the robot, which houses part of the cooling system, is open to allow for heat dissipation. The heat from the computers is transferred by the liquid in the cooling block, which is attached to the heat spreaders on each module. The liquid then moves through the radiator, which is cooled by the fans, before going back to the cooling block. In addition, the computing block is shock mounted on four cylindrical mounts that are used to absorb shocks and vibration.

shock-and-vibration damping standoffs. This makes it easier to properly evaluate the necessary damping requirements. When considering a damping solution, one needs to take into account the basic relationship between shock and vibration. That is, the solution has to be rigid enough to not cause too much vibration on the load but flexible enough to absorb shocks. Here, the focus is more on shock because, like regular laptops, the computers should be able to work despite the vibration that comes from reasonably rough terrains. In addition, the system uses solid-state hard drives (SSD), which have no moving parts and can withstand far more shock than their mechanical counterparts.

The natural rubber cylindrical mounts are selected over other options such as wire–rope isolators, rubber or silicone pads, and suspension springs because of their compactness. In addition, the height of the standoffs is easily adjustable by screwing together additional absorbers according to needs. Furthermore, one can change their shock absorption property by adding washers between two mounts if need be.

3.2.2. Cooling System

Because Beobot 2.0 is meant to be used both indoors and outdoors, we decided against an air-cooling system due to the possibility of the fans pushing dust into the exposed electronics inside, although air filters could have kept the dust out. However, the electronics would have to be placed in an area where air flow is well controlled, i.e., air must be drawn in and exhausted out only through the fans. This would have entailed a push-and-pull fan system and significant prototyping and rework of the mechanical system.

Therefore, we settled on a liquid-cooling solution. Moreover, as water has 30 times the amount of thermal conductivity and four times the amount of heat capacity as air (Callister, 2003), a liquid-cooling system is more effective in addition to being cleaner.

The liquid-cooling system, as shown in Figure 5, consists of the following components: cooling block, tubes, nozzles, radiator, two fans, liquid pump, reservoir, cooling liquid, a flowmeter, and a temperature sensor to monitor the system. Note that the system uses a cooling control board to provide power for the fans and the pump, as well as to take data from the flowmeter and temperature sensor.

The heat dissipated by the COM Express modules is first transferred to the liquid coolant through the processors' heat spreaders that are firmly pressed up against the top and bottom of the cooling block, which contains the coolant. We recommend using a high-performing, low-conducting, noncorrosive coolant for a maintenance-free system. The heat-carrying coolant first goes through the radiator, which has two fans pulling air through the radiator

surface. These fans are the devices that actively take the heat out of the system. Note that the radiator (and the fans) can be placed as far away from the processors as necessary. The liquid pump is connected to the system to ensure the flow of the liquid. Finally, a reservoir is included to add the coolant into the system and to take the air (bubbles) out of it.

4. SOFTWARE DESIGN

Our ultimate goal is to implement a fully autonomous embodied system with complete visual scene understanding. To do so, we lay the groundwork for a robot development environment (Kramer & Scheutz, 2002) that especially maximizes the multiple-processor hardware architecture. In addition it fulfills the primary objective in designing the software, viz., to be able to integrate and run computationally heavy algorithms as efficiently as possible. The advantage of using COM Express modules as a platform is that they can be treated as regular desktops. This allows the use of a Linux operating system in conjunction with C++ rather than some special-purpose environment. Note that, this way, the user can install any kind of Linux-compatible software tools that he/she prefers, not just the ones that we suggest below. Also, although this is not a true real-time system, it is quite adequate for our needs, with the control programs running reasonably fast and the robot responding in real time. In case a user would like to go with a real-time operating system, several Linux-based options and extensions are available (Politecnico di Milano, 2010; QNX Software Systems, 2010; Wind River, 2010; Xenomai, 2010).

To speed up the development of the complex algorithms mentioned above, we use the freely available iLab Neuromorphic Vision C++ Toolkit (Itti, 2009). The motivation for the toolkit is to facilitate the recent emergence of a new discipline, neuromorphic engineering, which challenges classical approaches to engineering and computer vision research. These new research efforts are based on algorithms and techniques inspired from and closely replicating the principles of information processing in biological nervous systems. Their applicability to engineering challenges is widespread and includes smart sensors, implanted electronic devices, autonomous visually guided robotics systems, prosthesis systems, and robust human–computer interfaces. Thus, the development of a neuromorphic vision toolkit helps provide a set of basic tools that can assist newcomers in the field with the development of new models and systems.

Because of its truly interdisciplinary nature, the toolkit is developed by researchers in psychology, experimental and computational neuroscience, artificial intelligence, electrical engineering, control theory, and signal and image processing. In addition, it aids in integration with other powerful, freely available software libraries such as Boost and OpenCV.

The project aims to develop next-generation general vision algorithms rather than being tied to specific environmental conditions or tasks. To this end, it provides a software foundation that can be used for the development of many neuromorphic models and systems in the form of a C++ library that includes classes for image acquisition, preprocessing, visual scene understanding, and embodied system control.

These systems can be deployed in a single machine or a distributed computing platform. We use the lightweight middleware ICE (Internet Communication Engine) via its C++ library bindings to facilitate intercomputer communication with a high-level interface that abstracts out low-level matters such as marshaling data and opening sockets. Sensors/devices, which are connected to a computer in the distributed system, are encapsulated as independent services that publish their data. Different systems can grab just the sensor outputs that they need by subscribing to that particular service. In addition, such a distributed system is fault tolerant as nonfunctional services do not bring down the whole system. We are also working on adding functionality to quickly detect nonresponding hardware and recover from failures by performing an ICE reconnection protocol, for example.

Another aspect to pay close attention to is the need for robust debugging tools for distributed systems that are provided by the toolkit as well as future applications. That is, we would like to know which modules in the system take the longest times, which ones send the largest amount of data, and how all these factors affect the overall system efficiency. Currently, the system has logging facilities for analysis after a testing run has taken place. What would be ideal is an online monitoring system.

In terms of hardware support, the toolkit has extensive source code available for interfacing sensors through different avenues. For example, Beobot 2.0 currently can connect to different types of cameras: USB, Firewire, or IP. Other devices that use a serial protocol should also be easily accommodated. In addition, it is important to note that the separation of hardware-related and algorithm-related code comes naturally. This allows the user to test most of the software in both the robot and our custom simulator (provided in the toolkit) without too many changes. Furthermore, the same robot cluster computing design is used for our robot underwater and aerial vehicles. We find that porting the algorithms to the other robots is done quite easily.

Table III lists all the vision- and robotic-related software capabilities provided by the toolkit.

5. TESTING AND RESULTS

We examine a few aspects of Beobot 2.0. The first is basic functionality such as power consumption, the cooling system, and mobility as it pertains to shock absorption. The power consumption testing shows the typical length

Table III. The vision toolkit features.

Features	Description	Available options
Devices	Interface code for various devices	Embedded systems/microcontrollers, joystick, keyboard, gyroscope, wii-mote, GPS, IMU (HMR3300, MicroStrain 3DM GX2), LRF (Hokuyo)
Robots	Control code for various robots	Scorbot robot arm, Evolution Robotics Rovio, Irobot Roomba, Beobot, Beobot 2.0, BeoSub Submarine, BeoHawk Quadrotor aerial robots, Gumbot for undergraduate introduction to robotics
Robotics algorithm	Modular mobile robotics algorithm	Localization, laser, and vision navigation (lane following, obstacle avoidance), SLAM
Distributed programming tools	Allows programs to communicate between computers	CORBA, Beowulf, ICE
Neuromorphic vision algorithms	Biologically plausible vision algorithms	Center-surround feature maps, attention/saliency (multithreaded, fixed point/integer), gist, perceptual grouping, contour integration, border ownership, focus of expansion, motion
Media	Access to various input media	mpeg, jpeg, cameras (USB, IP, IEEE1394 Firewire), audiovisual
Image processing	Various tools to manipulate images	Drawing, cut/paste, color operations [hue saturation value (HSV), RGB, etc.], statistical operations, shape transformation, convolutions, Fourier transform, pyramid builder, linear algebra/ matrix operation
Machine learning	Tools for pattern recognition training	K-nearest-neighbor, backpropagation neural networks, support vector machine, genetic algorithm
Object recognition	Visual object recognition modules	SIFT, HMAX

of operation given the amount of capacity of the batteries and the weight that the motors have to move and the eight computers that the batteries have to power. Beobot 2.0 has a power supply of 35 Ah $\times$ 24 V capacity from two 12-V SLA batteries in series. The robot is run with full-load computing by running a vision localization system (Siagian & Itti, 2009), explained in Section 5.2, while the robot is run around. In the testing, the cooling system is shown to drain about 1.8 A of the 24-V supply, whereas the gigabit switch and other sensors consume about 0.5 A. Each of the eight computers pulls up to 0.7 A during heavy use, and the motors pull 2 A when the robot is moving at about 1.61 km/h. The total comes up to 9.9 A in regular use, which corresponds to about 3.5 h of expected peak computation running time.

The good news is that Beobot 2.0 has two accessible power jacks located on its back, in the KVM board, as shown in Figure 4(b). By plugging in an auxilliary power source that stops its current flow when it detects another supply in the system, we can perform hot swapping to temporarily replace the SLA batteries. This prolongs the running time considerably, given that on-site system debugging occurs quite often. Consequently, the running time becomes actual testing time, without debugging time. This, for the most part, allows users to do research on site for the whole day and charge all night.

Table IV summarizes the results.

We then go into the usability of the system by reporting our experience implementing the nearness diagram (ND) navigation system (Minguez & Montano, 2004) in Section 5.1. Note that this section is included to show that the robot can move about an environment and is ready for use. We do not try to optimize the implementation to improve the performance. On the other hand, in Section 5.2, we describe our experiment performing three computationally intensive algorithms: the SIFT (Lowe, 2004) object recognition system, distributed visual saliency (Itti et al., 1998), and the robot vision localization system (Siagian & Itti, 2009). These computational speed/throughput experiments test the most critical aspect of the project's objectives. Given the complexity of having to implement a cluster of processors, we would like to see a good payoff for all our hard work.

5.1. Navigation Algorithm Implementation

In this section, we test the first algorithm to successfully run on Beobot 2.0, viz., the ND navigation algorithm (Minguez & Montano, 2004), which uses a laser range finder to build a proximity map around the robot and then searches this map for the navigable region closest to a goal location. A navigable region is a space or area that is at least as wide as the robot, thus enabling it to pass through. For example, the system's graphical user interface (GUI) display in Figure 6

Table IV. Beobot2.0 subsystem testings.

Subsystem	Tests	Results	Remarks
Liquid cooling	CPUs at full load at room temperature of 22°C	Average CPU temperature of 41°C	System is virtually maintenance free, although it consumes 1.8 A for the liquid pump; CPUs reach critical overtemperature within 15 min if liquid-cooling system is turned off
Mobility and shock absorption	Running the robot throughout the campus using RF controller at 2 m/s	Computers run smoothly without disconnection through several bumps and abrupt stops whenever the robot is too close to nearby pedestrians	We modify the motor controller code to properly ramp down when going to a complete stop
Battery consumption	Running the robot using the remote controller with all computers running full computations	Robot runs for 2.25 h before one CPU shuts down	Can prolong the testing time considerably by hot swapping batteries (there is a jack at the back of robot) during on-site debugging; consequently, the running time becomes actual testing time, without debugging time, and allows us to do research on site for the whole day

shows the robot's surroundings, divided into nine distinct regions.

The robot follows a series of binary decision rules that classify all situations into five mutually exclusive cases, which are summarized in Table V. Each case is associated with a corresponding movement action.

First, we define a security zone around the robot that is an area twice the robot's radius. In the GUI display (Figure 6), this zone is denoted by the light (yellow) center circle. If there are obstacles within the security zone (red dots within the circle in the figure), there are two cases to consider: whether there are obstacles on both sides of the robot or only on one side. In the former case, the robot tries to bisect this opening; in the latter case, it can move more freely to the open side. Note that the system considers only obstacles that are within 60 deg (between the two red lines in Figure 6) of the robot's direction of motion (blue line in the figure).

When there are no obstacles in the security zone, it considers three possible situations. If the goal location is in the navigable region, just go to it. If the goal location is not in the navigable region but the region is wide (only one obstacle on one of the sides), maneuver through the wide region, in the hope that there is a way to go to the goal region in the following time step. If goal location is not in the navigable region and the region is narrow (between two obstacles, one on each side), carefully move forward in the middle of the region. The overall resulting behavior is that the robot should continuously center itself between obstacles, while going to the goal.

To test this algorithm, only two of the available eight computers are needed. The laser range finder is plugged into one computer and the motor control board into the other. Additionally, a RC setup allows the user to change from autonomous to manual mode at the flick of a switch in case the robot is about to hit something or has been stuck in a corner for some time.

During implementation and debugging, a few notable features speed up the process. First, the 8-in. LCD screen allows users to observe the system states and action decisions as the robot is moving. Second, the use of a wireless USB keyboard and touch pad made it fairly easy to issue new commands while the robot was working. Last, but not least, taking the time to set up an intuitive GUI paid back dividends very quickly as it made it much easier to understand what was going on and how to fix the problems encountered.

The system is tested indoors, on a 20 × 24 ft empty area. We then occupy some of the regions with obstacles and test Beobot 2.0 to see whether it can navigate from one side of the environment and back. Figure 7 shows a snapshot of the environment setup for the experiment. In addition, some of the obstacle configurations are shown in Figure 8, with an example odometry trace overlaid on top.

There are nine different obstacle configurations and robot starting positions in the testing protocol. Each test was performed 10 times, with the robot's speed being the only variable parameter. We vary the speed between approximately 0.3 and 2.5 m/s. Table VI summarizes the results of each trial. For the most part, the navigation system

Table V. ND rules.

Number	Situation	Description	Action
1	Low safety 1	Only one side of obstacles in the security zone	Turn to the other side while maintaining the angle to the goal location
2	Low safety 2	Both side of obstacles in the security zone	Try to center between both side of obstacles and maintain the angle to the goal location
3	High safety goal in region	All obstacles are far from the security zone and goal	Directly drive toward the goal
4	High safety wide in region	All obstacles are far from the security zone but goal is not in this region	Turn half max angle away from closest obstacles
5	High safety narrow in region	All obstacles are far from the security zone and narrower region in the goal location	Center both side of closest obstacles

performs very well, with a 72% success rate. Here success is defined as the robot moving from its starting side of the environment to the other and back without touching any of the entities surrounding it.

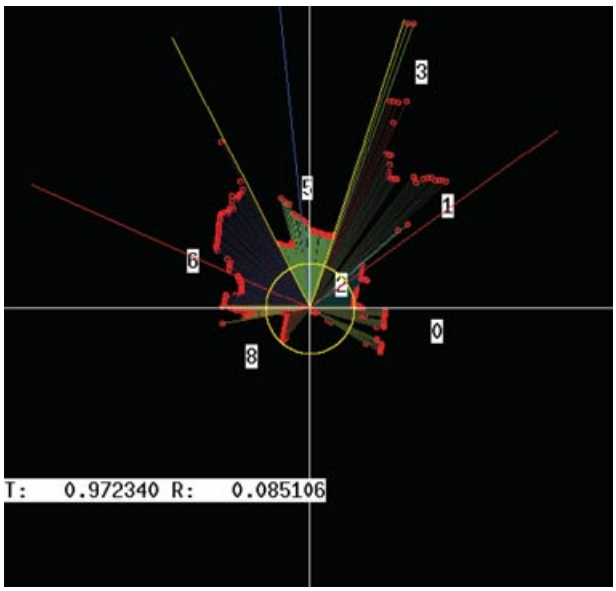


Figure 6. GUI of the ND navigation system. The system identifies nine (indexed from 0 to 8, note that 4 and 7 are cut off as they are drawn outside the frame of the GUI) different regions. The robot next direction of motion is denoted by the dark line next to label 5. The two lines next to the dark line delineate the boundaries of the navigable region. The red line indicates the directions 60 deg to the left and right of the robot's next direction. We also display the robot's translational and rotational motor command. Both of these numbers range from −1.0 to 1.0 (negative values indicate moving backward and counterclockwise rotation, respectively).

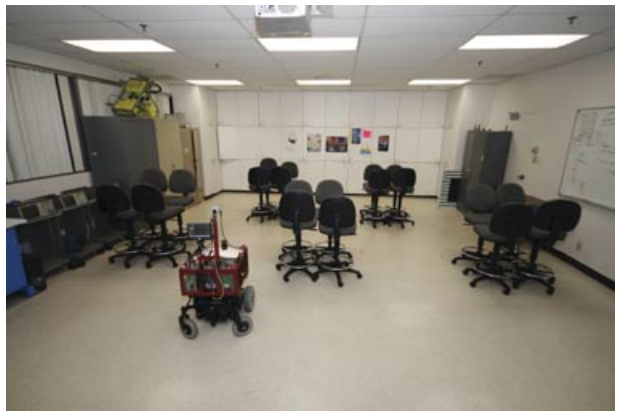
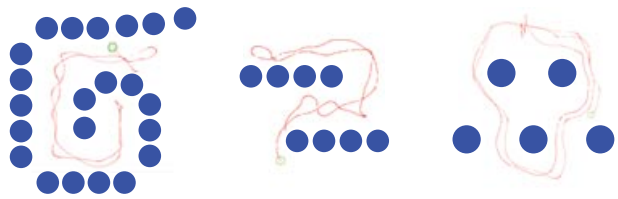


Figure 7. Snapshot of the constructed environment for ND navigation testing.



(a) Env1 (b) Env2 (c) Env3

Figure 8. Various environments for ND navigation testing with an example path that is taken by Beobot 2.0 using the ND navigation algorithm.

Of the total of 90 trials, 25 resulted in failures of some sort. Although this might seem excessive, it should be pointed out that the majority of the these collisions were of the type in which Beobot 2.0 only scraped an obstacle. This occurs whenever the robot has to turn sharply to avoid an obstacle, which causes its rear to scrape the obstacle. This is a minor problem that can be easily rectified by some simple

Table VI. Beobot 2.0 ND navigation testing.

End result	Occurrence	Percentage
Success	65	72.22
Scraping the obstacles	16	17.78
Stuck in corner or circles	7	7.78
Squarely hit an obstacle	2	2.22

control fix; e.g., when a turn is judged to be sharp, first back up a little.

There were two occasions when Beobot actually hit an obstacle head-on. This happened when the robot was running at its maximum speed. Under this circumstance, the latency of the system (the laser range finder throughput is 25 ms or 40 Hz, and processing is approximately the same duration as well) is simply too large to allow a timely reaction.

Finally, there were seven occasions when the robot became stuck in a corner or kept spinning in place because it kept alternating between left and right. The solution to this problem requires going beyond the simple reactive nature of the navigation system and figuring out what is globally optimal by integrating knowledge from localization or simultaneous localization and mapping (SLAM) algorithms.

5.2. Computational Capabilities

In this section, we characterize the computing platform by running three computationally intensive vision algorithms: SIFT (Lowe, 2004) object recognition system, the distributed visual saliency algorithm (Itti et al., 1998), and the biologically inspired robot vision localization algorithm (Siagian & Itti, 2009). These algorithms have a common characteristic in that their most time-consuming portions can be parallelized, whether it be distributing the feature extraction process (Section 5.2.2) or comparing those features to a large database (Sections 5.2.1 and 5.2.3). These parallel computations are then assigned to worker processes allocated at different computers in Beobot 2.0's cluster. Thus, we can fully test the computation and communication capabilities of the system.

5.2.1. SIFT Object Recognition System Test

As a first step in demonstrating the utility of our system in performing computationally intensive vision tasks, we implemented a simple keypoint matching system that is a very common component and performance bottleneck in many vision-based robotic systems (Se, Lowe, & Little, 2005; Valgren & Lilienthal, 2008). This task consists of detecting various interest points in an input image, computing a feature descriptor to represent each such point, and then searching a database of previously computed descriptors to determine whether any of the interest points in the current image have been previously observed. Once matches between newly observed and stored keypoints are found, the robot has a rough estimate of its surroundings and further processing such as recognizing its location or manipulating an object in front of it can commence.

Generally, the main speed bottleneck in this type of system is the matching between newly observed keypoints with the potentially very large database of previously observed keypoints. In the naive case this operation is $O(MN)$, where M is the number of newly observed keypoints and N is the number of keypoints in the database. However, this time can be cut to $O(M \log N)$ if the database of keypoints is stored as a KD-tree.

To test the efficacy of our cluster in speeding up such a task, we built a matching system in which a master node (called SIFT master) computes SIFT keypoints (Lowe, 2004) on an input image and then distributes these keypoints to a number of worker nodes (SIFT worker) for matching. Each of these workers is a separate process on the cluster located on either the same or a different machine as the master. Each worker contains a full copy of the database, stored as a KD-tree. Upon receiving a set of keypoints (different sets for each worker) from the master, a worker node compares each of them against its database and returns a set of unique IDs to the master representing the closest database match for each keypoint. Table VII describes the different types of modules in the system, and Figure 9 illustrates the flow of operation.

The database used for the experiment is composed of 905,968 SIFT keypoints obtained from HD footage (1,920 × 1,080 pixels) taken from an outdoor environment traversed by our robot. Each keypoint has 128 dimensions and eight bits per dimension. We vary the number of workers used to perform the keypoint matching from 1 to a maximum

Table VII. Beobot 2.0 SIFT object recognition time breakdown.

Operation	Description	Computation time
Input SIFT keypoint extraction	Extract SIFT keypoints from the input image; done by the master process	About 18 s
SIFT database matching	Match the input keypoints with the SIFT keypoints and return the results; done by the worker processes	16.25–353.22 s, depending on the number of workers utilized

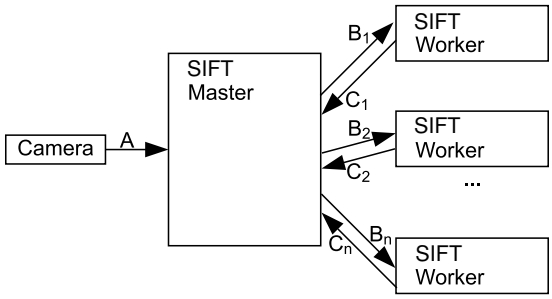


Figure 9. Flow of the distributed SIFT database matching algorithm denoted in increasing alphabetical order and referred below in parenthesis. First the camera passes in the high-definition (1,920 × 1,080 pixel) frame to the SIFT master module (A). This module takes about 18 s to extract the SIFT keypoints from the input image before sending them to the SIFT worker processes utilized (denoted as B_i, i being the total number of workers). Depending on the number of workers, each takes between 16.25 and 353.22 s to return a match to the SIFT Master (C_i).

Table VIII. Beobot 2.0 SIFT database matching algorithm testing results.

Number of workers	Processing time (s/frame)	Standard deviation (ms/frame)
1	353.2194	57.8369
2	193.6876	58.1703
3	130.8932	39.6815
4	95.5375	27.8618
5	95.3156	34.4357
6	57.5809	12.2352
7	47.1917	10.6182
8	40.2749	9.9586
9	37.2474	8.3234
10	29.8436	6.6259
11	37.7632	15.9283
12	18.9525	6.2032
13	20.9672	6.1088
14	25.3063	6.8077
15	16.2541	5.7455

of 15 (because a total of 16 cores are available). Figure 10 illustrates the allocations of the modules. Table VIII and Figure 11 record the time required to process each frame, plotted against the number of workers.

Table VIII shows a total decrease of 21.73 times (from 353.22 to 16.25 s) in per frame processing time between 1 and 15 workers. Here, that the improvement goes beyond 15-folds is, we believe, because of memory paging issues that arise when dealing with the large messages necessary when using a small number of nodes.

Figure 11 also shows that while diminishing returns are achieved after 11 nodes, there is still a significant performance improvement by the utilization of parallel workers in the cluster. Although not all algorithms are as easily parallelized as this example, this experiment shows that a very common visual localization front end can indeed be parallelized and the benefits for doing so are significant.

5.2.2. Distributed Visual Saliency Algorithm Test

One of the capabilities that is important in a robot is object collection. Here, a key task to perform is object recognition, usually from an image. There are times when the object may be small, or placed in a cluttered environment. This is when an algorithm such as the saliency model (Itti et al., 1998) can be quite useful. The term saliency is defined as a measure of conspicuity in an image, and by estimating this characteristic for every pixel in the image, parts of it that readily attract the viewer's attention can be detected. Thus, instead of blindly performing an exhaustive search throughout the input image, the saliency model can direct the robot to the most promising regions first. We can then equip the robot with a high-resolution camera to capture all the details of its surroundings. Furthermore, because of Beobot 2.0's powerful computing platform, saliency processing in such a large image in a timely manner becomes feasible.

To compute the salience of an image, the algorithm (Itti et al., 1998) first computes various raw visual cortex features that depict visual cues such as color, intensity, orientation (edges/corners), and flicker (temporal change in intensity). Here we have multiple subchannels for each domain: 2 color opponencies (red–green and blue–yellow center-surround computation), 1 intensity opponency (dark–bright), 12 orientation angles (increments of 15 deg), and 1 for flicker. That is a total of 16 subchannels, each producing a conspicuity map, which are then combined to create a single saliency map.

Because the computations in each subchannel are independent, they can be easily distributed. And so we use the algorithm to show how having many cores in a robot can alleviate such a large computational demand. For the experiment, we set up 1 master process and 1–15 worker processes to calculate the saliency of images of 4,000 × 4,000 pixels in size, for 100 frames. The master process takes approximately 100 ms to preprocess the input image before sending the jobs to the workers. The jobs themselves take up to 100 ms to finish for the color, intensity, and flicker subchannels and up to 300 ms for the 12 orientation subchannels. Finally, the conspicuity map recombination takes less than 10 ms. Table IX summarizes the running times of individual parts of the system, and Figure 12 illustrates the flow of the algorithm. In addition, Figure 13 shows the actual allocations of all the processes at which computer the modules are run.

The results that we obtained from this experiment can be viewed in Table X and are graphed in Figure 14. As we

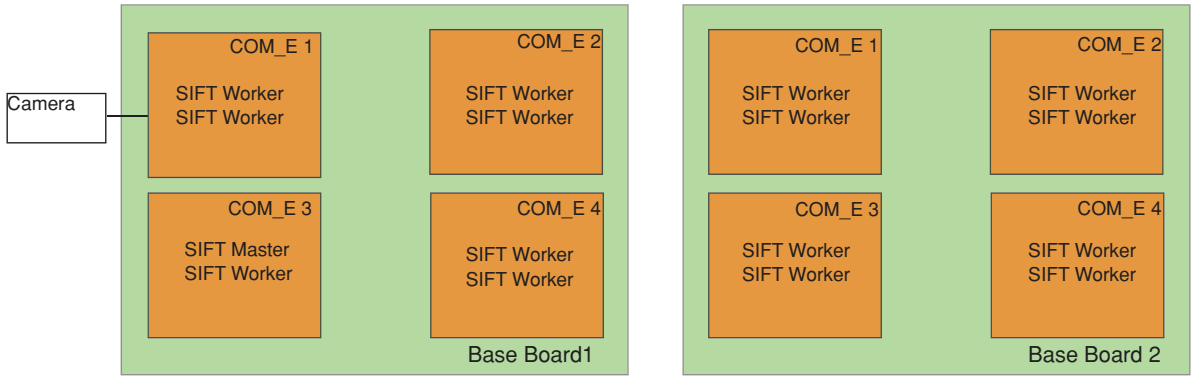


Figure 10. Allocation of the different programs of the distributed SIFT database matching algorithm in Beobot 2.0. The SIFT master module is run on one of the cores in computer COM_E1, and the various SIFT worker modules are allocated throughout the cluster.

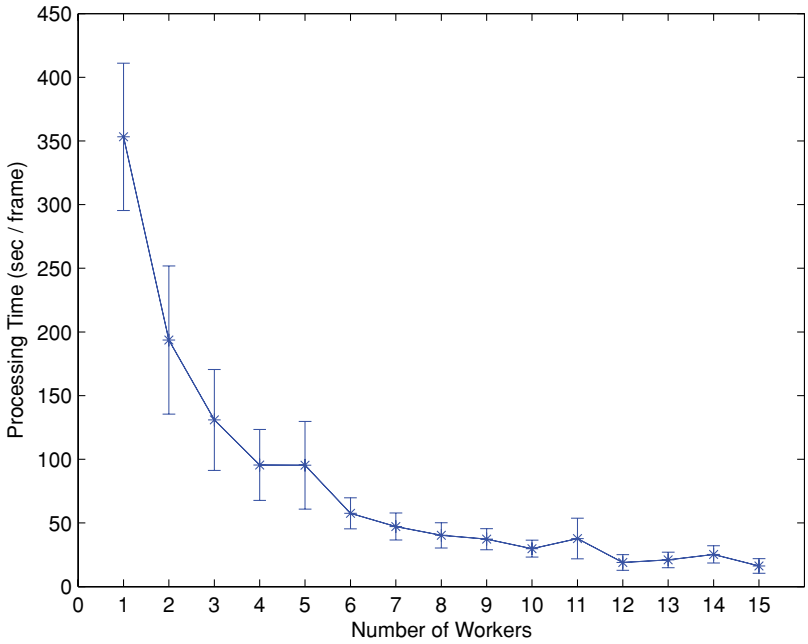


Figure 11. Results for SIFT database matching testing on Beobot 2.0.

can see, the processing time drops as we continue to add workers to the system. Quantitatively, the processing time reduction comes reasonably close to the expected value, at least early on. For example, if using one worker, the processing time is 3,456.50 ms, then using two workers should take half the time, 1,728.25 ms, which is comparable to the actual time of 1,787.69 ms. This is usually the case for a straightforward distributed processing in which there are no dependencies between the processes.

Another point of comparison is that we would like to gauge the improvement using the full cluster with what would be equivalent to a standard quad core system. Thus, we compare the usage of 3 worker nodes against all 15 nodes. We see a slight drop in improvement to 3.55 (from 1,249.68 to 352.26 ms). This slowdown is primarily attributed to network congestion, as we are shuffling large images around. Furthermore, if we compare the running time of 1 worker (3,456.5 ms) with 10 times the running time of 10 workers (478.33 ms $\times$ 10 $=$ 4,783.3 ms), there seems to be a lot of added time. And so, as we add more and more workers, we expect to eventually hit a point of diminishing returns. A lesson to be taken here is that we should consider not only how to divide the task and properly balance job allocation but also how large the data set (or the total communication cost) is that needs to be distributed for each assigned job.

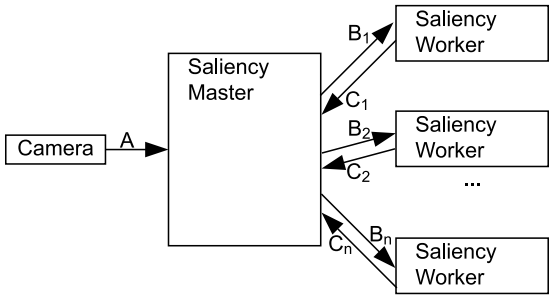


Figure 12. Flow of the distributed saliency algorithm denoted in increasing alphabetical order and referred below in parenthesis. First the camera passes in the high-resolution 4,000 × 4,000 pixel image to the SalMaster module (A). SalMaster preprocesses the image, which takes 100 ms, before sending out the image to various subchannel SaliencyWorker processes (denoted as B_i, i being the total number of workers). The color, intensity, and flicker subchannels take up to 100 ms, and the orientation subchannels take up to 300 ms. These results are then recombined by SalMaster (C_i), and this takes less than 10 ms.

5.2.3. Biologically Inspired Robot Vision Localization Algorithm Test

For the third computational test, we utilized the vision localization algorithm by Siagian and Itti (2009). It relies on matching localization cues from an input image with a large salient landmark database obtained from previous training runs to capture the scenes from the target environment under different lighting conditions.

The algorithm first computes the same raw visual cortex features that are utilized by the saliency algorithm (Itti et al., 1998). It then uses these raw features to extract gist information (Siagian & Itti, 2007), which approximates holistic aspects and the general layout of an image, to coarsely locate the robot in a general vicinity. In the next step, the system then uses the same raw features to isolate the most salient regions in the image and compare them with the salient regions stored in the landmark database to refine its whereabouts to a metric accuracy. The actual matching

Table IX. Beobot 2.0 distributed saliency algorithm time breakdown.

Module	Description	Computation time (ms)
Input image preprocessing	Computes luminance and red–green and blue–yellow color opponency maps to be sent to the worker processes; done by the master process	100
Conspicuity map generation	Performs center-surround operations in multiple scales to produce a conspicuity map for each subchannel	300–3,900; depends on the number of workers utilized
Saliency map generation	Combines all the conspicuity maps returned by the workers to a single saliency map; done by the master process	10

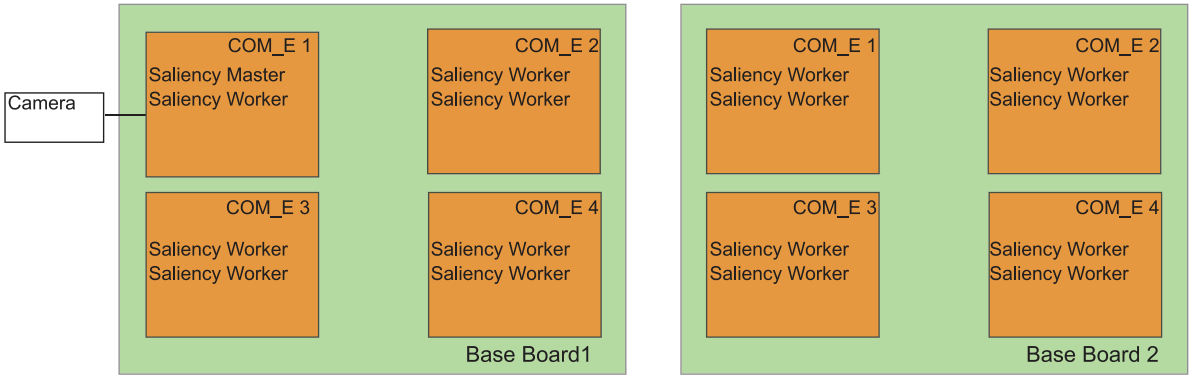


Figure 13. Allocation of the different programs of the distributed saliency algorithm in Beobot 2.0. The saliency master module is run on one of the cores in computer COM_E1, and the various saliency worker modules are allocated throughout the cluster.

Table X. Beobot 2.0 visual saliency algorithm testing results.

Number of workers	Processing time (ms/frame)	Standard deviation (ms/frame)
1	3,456.50	108.904
2	1,787.69	491.056
3	1,249.68	528.787
4	979.14	374.978
5	733.85	359.345
6	629.24	387.554
7	571.79	429.028
8	526.77	288.188
9	498.49	441.441
10	478.33	482.481
11	452.13	290.235
12	436.75	253.642
13	375.59	299.921
14	362.72	195.126
15	352.26	332.128

between two regions itself is done using SIFT features (Lowe, 2004). Of the different parts of the algorithm, the salient region recognition process takes the longest time. However, the computations performed by this module are parallelizeable by dispatching workers to compare particular parts of the database. Aside from the parallel searches, there are two other processes whose jobs are to extract gist and saliency features from the input image and a master process that assigns jobs and collects results from all landmark database search worker processes.

The gist and saliency extraction process, which operates on 160 × 120 size images, takes 30–40 ms to complete per frame and has to be run first. The images are placed in the computer that is connected to the camera. For this experiment, however, we are running off of previously collected data, without running the motors. Note that because the information being passed around consists of a few small salient regions (about five regions of 40 × 30 pixels, on average), only a small amount of time (4–5 ms) is spent on data transfer (using the ICE protocol) through the gigabit Ethernet network.

We then run the master search process, which takes about 50–150 ms (depending on the number of landmarks in the database) to create a priority queue for ordering landmark comparisons from most to least likely using saliency, gist, and temporal cues. For example, in the gist-based prioritization, if the gist features of the input image suggest that the robot location is more likely to be in a certain vicinity, we compare the incoming salient region with the stored regions found near that place. This prioritization improves the system speed because we are trying to find only a first match, which halts the search once it is found, and not the best match, which requires an exhaustive search through the entire database.

After these two processes are done, we can then dispatch the landmark search processes in parallel. For testing purposes, we use one, two, four, and eight computers to examine the increase in overall system speed. Noting that

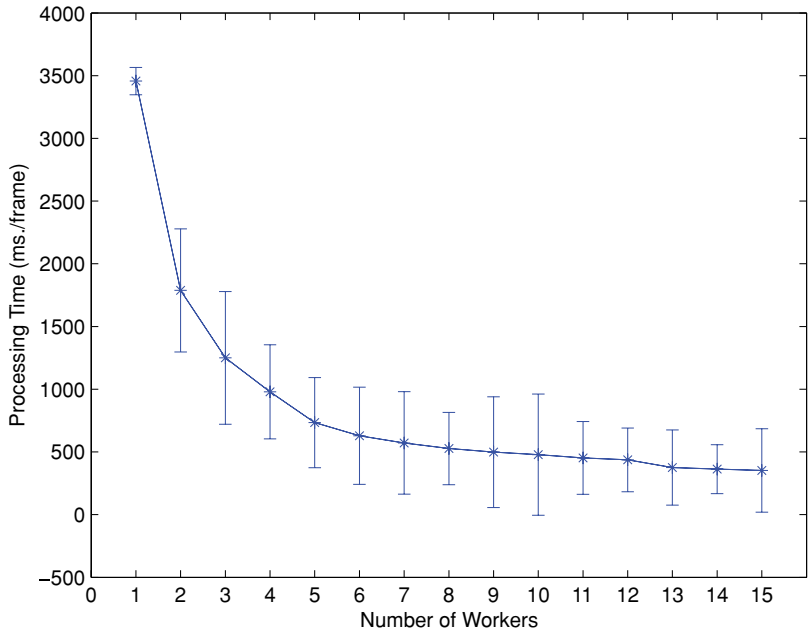


Figure 14. Results for saliency algorithm testing on Beobot 2.0.

Table XI. Beobot 2.0 localization system time breakdown.

Module	Description	Computation time
Gist and saliency	Computes various raw visual cortex features (color, intensity, and orientation) from the input image for gist and saliency feature extraction	30–40 ms
Localization master	Creates a priority job queue (to be sent to the workers) for ordering landmark database comparisons from most to least likely using saliency, gist and temporal cues; it also collects the search results from all search workers	50–150 ms; depends on the size of the database
Localization worker	Compares the incoming salient region with the stored regions based on the prioritization order; the search halts once the first positive match is found	300–3000 ms; depends on the size of the database and the number of workers utilized

there are two cores in each computer, we dispatch 2, 4, 8, and 16 landmark database worker processes, respectively. The localization master then collects the match results to deduce the robot's most likely location given the visual evidence. This final step takes less than 5 ms.

Table XI summarizes the various processes, Figure 15 shows the program allocation, and Figure 16 illustrates the flow of the algorithm.

We test the system on the same data set as that used in Siagian & Itti (2009), which depicts a variety of visually challenging outdoor environments from a building complex (ACB) to parks full of trees (AnFpark) to an open area (FDFpark). The database information for each site in their respective rows, can be found in Table XII, and the images can be viewed in Figure 17. The table shows the number of training sessions, each of which depicts a different lighting condition in the outdoor environments. This is one of the reasons why the database is so large. The table also introduces the term salient region (Siagian & Itti, 2009), denoted as SRegs, which is different from a landmark. A landmark is a real entity that can be used as a localization cue, whereas a salient region is evidence obtained at a specific time. Thus there are, on average, about 20 salient regions to depict a landmark to cover different environmental conditions.

The results are shown in Table XIII. Here, we examine the processing time per frame, the localization error, and salient regions found per frame. As we can see from the table, for each site there is always a decrease in processing time per frame as we increase the number of computers. At the same time, generally, there is an increase in accuracy in two of the three sites as the number of computers is increased. The reason for this is that the localization

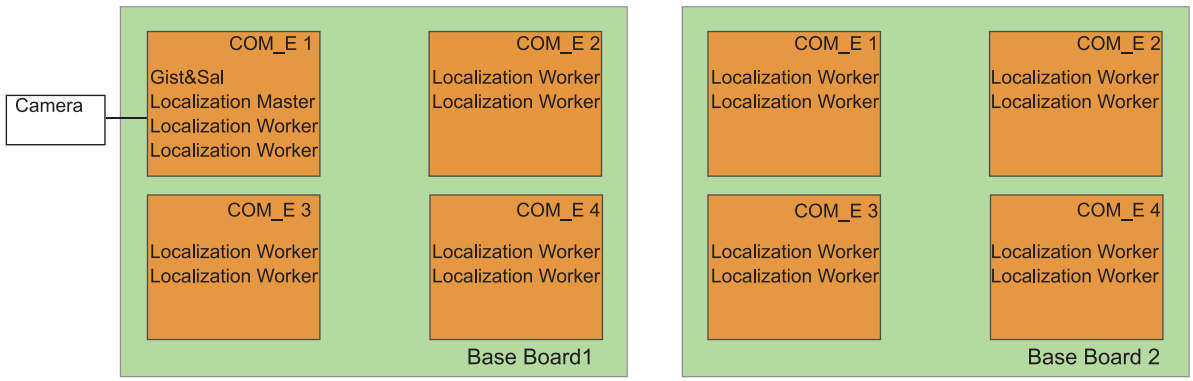


Figure 15. Allocation of the different programs of the localization system in Beobot 2.0. The gist and saliency extraction (GistSal) and localization master modules are allocated computer COM_E1, and the various localization worker modules are assigned to cores throughout the cluster. Note that there are also two worker modules in COM_E1. This is because they run only when GistSal and localization master modules do not, and vice versa.

Table XII. Beobot 2.0 vision localization system testing.

Environment	Number of training sessions	Number of testing frames	Number of lmk	Number of S. Regs	Number of S. Reg/Lmk
ACB	9	3,583	1,501	19.79	29,710
AnFpark	10	6,006	4,664	17.69	82,502
FDFpark	11	8,823	4,808	18.86	90,660

Table XIII. Beobot 2.0 vision localization system testing results.

	ACB			AnF			FDF		
Number of computers	Time	Err (m)	S. Reg found/frame	Time	Err (m)	S. Reg found/frame	Time	Err (m)	S. Reg found/frame
1	1,181.60	2.21	2.34/4.89	2,387.87	2.27	2.73/4.98	3,164.96	4.30	2.51/4.78
2	711.93	1.70	2.38/4.89	1,495.23	2.31	2.76/4.98	1,909.01	4.36	2.51/4.78
4	499.18	1.13	2.48/4.89	1,000.66	2.36	2.81/4.98	1,201.90	4.04	2.55/4.78
8	421.45	1.26	2.57/4.89	794.31	2.38	2.94/4.98	884.74	4.08	2.60/4.78

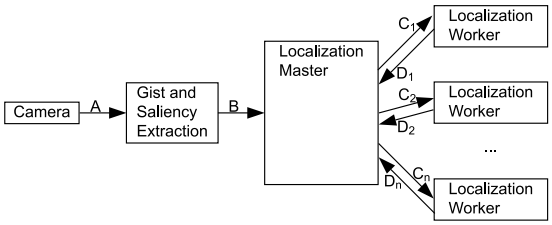


Figure 16. Flow of the localization system denoted in increasing alphabetical order and referred below in parenthesis. First the camera passes in a 160 × 120 pixel image to the gist and saliency extraction module (A), which takes 30–40 ms, before sending out the localization master module (B). This module then allocates search commands/jobs in a form of priority queue to be sent to a number of localization workers (C_i, i being the total number of workers) to perform the landmark database matching. A search command job specifies which input salient region is to be compared to which database entry. This takes 50–150 ms, depending on the size of the database. The results are then sent back to the localization master (D_i) to make the determination of the robot location given the visual matches. The last steps takes less than 10 ms.

algorithm itself behaves differently as more and more resources are provided, in that it tries to optimize between the speed of computation and the accuracy of the results. Consequently, the running time analysis is not as straightforward. That is, we cannot just look at the nonlinearity of the relationship between the number of computers and processing time, stating doubling the number of computers does not halve the processing time, and say that the algorithm does not take advantage of the available computing efficiently.

As we explained earlier, the Siagian and Itti (2009) localization system orders the database landmarks from the most to the least likely to be matched. This is done by using other contextual cues (such as gist features, salient feature vectors, and temporal information) that can be computed much quicker than in the actual database matching process. The effect of this step is that it gives robot systems with limited computing resources the best possible chance to match the incoming salient regions. In addition, there is also an early-exit strategy that halts the search if the following conditions are met:

- Three regions are matched.
- Two regions are matched and 5% of the queue has been processed since the last match.
- One region is matched and 10% of the queue has been processed since the last match.
- No regions are matched and 30% of the queue has been processed.

This policy is designed to minimize the amount of unnecessary work when it is obvious that a subsequent match is very unlikely to be found. However, together with the increase in the number of workers, this policy actually creates a slightly different behavior.

First, there is a difference between the number of jobs processed by a one-worker setup compared to a multiple-worker setup. In the former setup, the localizer master process assigns a job, waits until the worker is done, and then checks whether any of the early-exit conditions are met before assigning another job. In the multiple-worker case, the master assigns many jobs at the same time and much more frequently. This increases the possibility of a match,

Figure 17. Examples of images in the ACB (first row), AnFpark (second row), and FDFpark (third row).

as demonstrated by the increase in the number of salient regions found in Table XIII. In turn, this slows the running time by prolonging the search process by 5%, 10%, or even 15% (in a compound case) of the queue, depending on which early-exit conditions are invalidated.

On the other hand, however, the higher number of matches found can also increase the accuracy of the system, but not always. As we can see in Table XIII, there is a small but visible adverse effect of letting the search go too long (most clearly in the AnF site). This is because the longer the search process, the more likely that a false-positive is discovered as the jobs lower in the priority queue are in lesser agreement with other contextual information. Furthermore, this is also reflected by the fact that a lot of the salient regions are found early in the search as the numbers do not increase significantly as we add more computers. For example, in the ACB site, compare the salient region found using one computer (2.34) with using eight (2.57).

From the table, we estimate that, for these environments, four computers appears to be the optimum number. Note that the localization system does not have to be real time, but being able to come up with a solution within seconds, as opposed to a minute, is essential because longer durations would require the robot to stop its motor and stay in place. This is what we are able to do with Beobot 2.0. In the full setup, the localization system is going to be run in conjunction with a salient region tracking mechanism, which keeps track of the regions while it is being compared with the database while still allowing the robot to move freely as long as the region is still in the field of view. If we use just four of the computers for localization, the others can be used for navigational tasks such as lane finding, obstacle avoidance, and intersection recognition, thus making the overall mobile robotic system real time. Currently, we have a preliminary result of a system that localizes and navigates autonomously in both indoor and outdoor environments, reported in Chang, Siagian, and Itti (2010).

6. DISCUSSION AND CONCLUSIONS

In this paper, we have described the design and implementation of an affordable research-level mobile robot platform equipped with a computing cluster containing eight dual-core processors for a total of 16 2.2-GHz CPUs. With such a powerful platform, we can create highly capable robotic applications that integrate many complex algorithms, use many different advanced libraries, and utilize large databases to recall information. In addition, by using the COM Express form-factor industry standard, the robot is able to stave off obsolescence for a longer period due to the ability to switch COM modules and upgrade to the latest processor and computer technology.

Furthermore, by implementing our own robot, we have demonstrated a cost-effective way to build such a computationally powerful robot. For more information on the cost breakdown, please refer to Siagian et al. (2009). The trade-off, of course, is in development time and effort. In our lab, we have had two people working full time on this project, with a few others helping out here and there. The total design and implementation time has been 18 months from conception to realization. We have had to think about many issues, no matter how small or seemingly trivial, in order to ensure that no major flaws are introduced into the design that can become showstoppers down the road. However, given that we now have the final design (Siagian et al., 2009), the implementation of a second robot ought to be relatively straightforward and much quicker, on the order of 2–3 months.

One might wonder why we would go to such as extraordinary effort to build such a complex hardware system when there may well be an easier alternative. For example, why not simply stack eight laptops on a mobile platform connected with Ethernet cables? We believe that with such an approach, it would be hard to isolate the computers from the elements. Cooling, for instance, would have to be done in two steps. First, the internal laptop fans would blow hot air out to a waterproof inner compartment of the

robot body. Then, we would need another set of fans, fitted with filters to drive the air in and out of the robot body. Furthermore, we would still have to create space to place all the other devices (for example, sensors and motor driver), along with power connections that also have to supply the main computers. The resulting shear numbers of cables would easily make the system unwieldy and unappealing.

In our custom design, however, wires and cabling, which are often a source of connection failures, have been kept to a minimum thanks to the printed circuit board (PCB) design directives. Additionally, the liquid-cooling system is well sealed and runs smoothly every time we run the robot. In terms of maintenance, we use the robot every day and have found it to be quite trouble free. The wall plug-in battery charging system, for one, makes it convenient to charge the robot at night before going home. Finally, we would like to add that, although in this paper we are presenting a terrestrial system, the same technology has been applied to an underwater robot (USC Robotics, 2009), where dimensions and weights become critical factors and modifying COTS (commercial off the shelf) systems may not be feasible.

Nonetheless, with the benefit of hindsight, there are some things we would have liked to improve upon. One is easier access to various electronic components inside the robot body. For example, four of the COM Express modules are placed underneath the cooling block structure, and taking them out for repair can be somewhat difficult. This is the price we pay for designing such a highly integrated system. Another problem is managing many computers. In an application that requires all eight of Beobot 2.0's computers, we have to compile and run many programs in parallel with certain ordering constraints. In addition, we also have to properly allocate where each program should be run on which computers, so that there are no computers that are idling while others are overloaded. Although these issues cannot always be avoided, some forethought and automation via appropriate scripting can help. Frameworks such as MOSIX or the Scyld Beowulf system are available to aid this process, which is to be tested in the future on our robot.

In the end, we believe that our primary contribution is that Beobot 2.0 allows for a class of computationally intensive algorithms that may need to access large-sized knowledge databases, operating in large-scale outdoor environments, something that previously may not have been feasible on commercially available robots. In addition, it also enables researchers to create systems that run several of these complex modules simultaneously, which is exactly what we are currently working on in our lab. That is, we would like to run the localization system (Siagian & Itti, 2009), vision-based obstacle avoidance, and lane following (Ackerman & Itti, 2005) together. We also are planning to add components such as SLAM and human/robot interaction. The long-term goal is to make available plenty of predefined robotic components that can be reused to speed up future project developments.

Subsequently, the problem that we foresee is managing these diverse capabilities. We have to make sure that there are enough resources to work with and give priority to the most important and reliable subsystems in solving the task at hand as well as identifying dangers that threaten the livelihood of the robot. We hope that through our contribution of implementing an economical but powerful robot platform, we can start to see more of the type of complete systems that are needed to thrive in a real-world environment.

ACKNOWLEDGMENTS

This work was supported by the Defense Advanced Research Projects Agency (government contract no. HR0011-10-C-0034), the National Science Foundation (CRCNS grant number BCS-0827764), the General Motors Corporation, and the Army Research Office (grant number W911NF-08-1-0360). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

REFERENCES

AAI Canada, Inc. (2009). AAI Canada, Inc.—Intelligent robots—Khepera II. http://www.aai.ca/robots/khep_2.html. Accessed October 15, 2009.

Ackerman, C., & Itti, L. (2005). Robot steering with spectral image information. IEEE Transactions on Robotics, 21(2), 247–251.

Altium Limited. (2009). Altium—Next generation electronics design. http://www.altium.com. Accessed July 15, 2009.

American Honda Motor Co., Inc. (2009). Asimo—The world's most advanced humanoid robot. http://asimo.honda.com/. Accessed July 15, 2009.

Bay, H., Tuytelaars, T., & Gool, L. V. (2006, May). SURF: Speeded up robust features. In ECCV, Graz, Austria (pp. 404–417).

Brooks, R. A. (1986). A robust layered control system for a mobile robot. IEEE Transactions on Robotics and Automation, 2(1), 14–23.

Callister, W. (2003). Materials science and engineering—An introduction. Hoboken, NJ: Wiley.

Carnegie Mellon University Robotics Institute. (2009). LAGR robot platform. http://www.rec.ri.cmu.edu/projects/lagr/description/index.htm. Accessed July 15, 2009.

Chang, C.-K., Siagian, C., & Itti, L. (2010, October). Mobile robot vision navigation & localization using gist and saliency. In Proceedings IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Taipei, Taiwan.

Chung, D., Hirata, R., Mundhenk, T. N., Ng, J., Peters, R. J., Pichon, E., Tsui, A., Ventrice, T., Walther, D., Williams, P.,

& Itti, L. (2002, November). A new robotics platform for neuromorphic vision: Beobots. In Lecture Notes in Computer Science (vol. 2525, pp. 558–566).

COM Express Extension. (2009). COM Express Extension—Consortium. http://www.comexpress-extension.org/specs/specs.php. Accessed October 15, 2009.

Dimension Engineering LLC. (2009). Sabertooth 2X25 regenerative dual motor driver. http://www.dimensionengineering.com/Sabertooth2X25.htm. Accessed July 15, 2009.

ETX Industrial Group. (2009). ETX Industrial Group—Consortium. http://www.etx-ig.org/consortium/consortium.php. Accessed July 15, 2009.

Evolution Robotics, Inc. (2009). Evolution robotics development platform and OEM solutions for robot software navigation technology, object recognition. http://www.evolution.com. Accessed July 15, 2009.

Finio, B., Eum, B., Oland, C., & Wood, R. J. (2009, October). Asymmetric flapping for a robotic fly using a hybrid power control actuator. In IROS, St. Louis, MO.

Fox, D., Burgard, W., Dellaert, F., & Thrun, S. (1999, July). Monte Carlo localization: Efficient position estimation for mobile robots. In Proceedings of Sixteenth National Conference on Artificial Intelligence (AAAI'99), Orlando, FL.

Fox, D., Burgard, W., Kruppa, H., & Thrun, S. (2000). A probabilistic approach to collaborative multi-robot localization. AutonomousRobots, 8(3), 325–344.

He, R., Prentice, S., & Roy, N. (2008, May). Planning in information space for a quadrotor helicopter in GPS-denied environments. In ICRA2008, Pasadena, CA.

Heitz, G., Gould, S., Saxena, A., & Koller, D. (2008, December). Cascaded classification models: Combining models for holistic scene understanding. In Advances in Neural Information Processing Systems (NIPS 2008), Vancouver, BC, Canada.

Hokuyo Automatic Co., Ltd. (2009). Photo sensor—PRODUCTS. http://www.hokuyo-aut.jp/02sensor/index.html#scanner. Accessed July 15, 2009.

iRobot Corporation. (2009a). PackBot. http://www.irobot.com/sp.cfm?pageid=171. Accessed July 15, 2009.

iRobot Corporation. (2009b). Roomba vacuum cleaning robot. http://store.irobot.com / category / index.jsp?categoryId=3334619&cp=2804605. Accessed July 15, 2009.

iRobot Corporation. (2010). SeaGlider. http://www.irobot.com/sp.cfm?pageid=393. Accessed January 15, 2010.

Itti, L. (2009). iLab Neuromorphic Vision C++ Toolkit (iNVT). http://ilab.usc.edu/toolkit/. Accessed July 15, 2009.

Itti, L., Koch, C., & Niebur, E. (1998). A model of saliency-based visual attention for rapid scene analysis. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20(11), 1254–1259.

Ituner Networks Corp. (2009). PicoPSU-80-WI-32V power supply. http://www.mini-box.com/PicoPSU-80-WI-32V?sc=8&category=981. Accessed July 15, 2009.

Kontron. (2007). Design guide for ETXexpress carrier boards. Poway, CA: Kontron.

Kontron. (2009). ETXexpress-MC. http://us.kontron.com / products / computeronmodules / com+express/etxexpress/etxexpressmc.html. Accessed July 15, 2009.

Kramer, J., & Scheutz, M. (2002). Development environments for autonomous mobile robots: A survey. Autonomous Robots, 22(2), 101–132.

Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2), 91–110.

Maes, P., & Brooks, R. A. (1990, July–August). Learning to coordinate behaviors. In AAAI, Boston, MA (pp. 796–802).

Major's Mobisist. (2009). Major's Mobisist :: Power Wheelchairs :: Liberty 312. http://www.movingwithdignity.com/product.php?productid=16133. Accessed July 15, 2009.

MicroStrain, Inc. (2009). 3DM-GX2:: MicroStrain, AHRS Orientation Sensor. http://www.microstrain.com/3dm-gx2.aspx. Accessed July 15, 2009.

Mikolajczyk, K., & Schmid, C. (2005). A performance evaluation of local descriptors. IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(10), 1615–1630.

Minguez, J., & Montano, L. (2004). Nearness diagram (ND) navigation: Collision avoidance in troublesome scenario. IEEE Transactions on Robotics and Automation, 20(1), 45–59.

MobileRobots, Inc. (2009). Seekur unmanned ground vehicle. http://www.mobilerobots.com/Seekur.html. Accessed July 15, 2009.

MobileRobots, Inc. (2009). The High Performance All-Terrain Robot. http://www.activrobots.com/ROBOTS/p2at.html. Accessed July 15, 2009.

National Institute of Standards and Technology. (2009). Why are there no volume Li-ion battery manufacturers in the United States? http://www.atp.nist.gov/eao/wp05-01/append-4.htm. Accessed July 15, 2009.

PNI Sensor Corporation. (2009). PNI Sensor Corporation—Sensors modules—All products—MicroMag 3 : 3-Axis magnetometer. http://www.pnicorp.com/products/all/micromag-3. Accessed July 15, 2009.

Politecnico di Milano. (2010). RTAI—the RealTime Application Interface for Linux from DIAPM. https://www.rtai.org/. Accessed July 15, 2010.

Pomerleau, D. (1993). Knowledge-based training of artificial neural networks for autonomous robot driving. Robot learning (pp. 19–43). New York: Springer.

QNX Software Systems. (2010). QNX realtime RTOS—Middleware, development tools, realtime operating system software and services for superior embedded design. http://www.qnx.com/. Accessed July 15, 2010.

Qseven Standard. (2009). Qseven: About Qseven. http://www.qseven-standard.org/. Accessed October 15, 2009.

Quigley, M., & Ng, A. (2007, July). Stair: Hardware and software architecture. In AAAI 2007 Robotics Workshop, Vancouver, BC, Canada.

Salichs, M. A., Barber, R., Khamis, A. M., Malfaz, M., Gorostiza, J. F., Pacheco, R., Rivas, R., Corrales, A., Delgado, E., & Garca, D. (2006, June). Maggie: A robotic

platform for human–robot social interaction. In International Conference on Robotics, Automation, and Mechatronics (RAM 2006), Bangkok Thailand.

Se, S., Lowe, D. G., & Little, J. J. (2005). Vision-based global localization and mapping for mobile robots. IEEE Transactions on Robotics, 21(3), 364–375.

Segway, Inc. (2009). Segway—Business—Products & solutions—Robotic mobility platform (RMP). http://www.segway.com/business/products-solutions/robotic-mobility-platform.php. Accessed July 15, 2009.

SensComp, Inc. (2009). Mini Sensors. http://www.senscomp.com/minis.htm. Accessed July 15, 2009.

Siagian, C., Chang, C. K., Voorhies, R., & Itti, L. (2009). Beobot 2.0. http://ilab.usc.edu/wiki/index.php/Beobot_2.0. Accessed July 15, 2009.

Siagian, C., & Itti, L. (2007). Rapid biologically-inspired scene classification using features shared with visual attention. IEEE Transactions on Pattern Analysis and Machine Intelligence, 29(2), 300–312.

Siagian, C., & Itti, L. (2009). Biologically inspired mobile robot vision localization. IEEE Transactions on Robotics, 25(4), 861–873.

SolidWorks Corp. (2009). SolidWorks 3D CAD Design Software. http://www.solidworks.com/. Accessed July 15, 2009.

Sony Entertainment Robot Europe. (2009). Aibo. http://support.sony-europe.com/aibo/. Accessed January 15, 2009.

Thrun, S., Fox, D., & Burgard, W. (1998). A probabilistic approach to concurrent mapping and localization for mobile robots. Machine Learning, 31, 29–53.

Thrun, S., Fox, D., Burgard, W., & Dellaert, F. (2000). Robust Monte-Carlo localization for mobile robots. Artificial Intelligence, 128(1–2), 99–141.

Thrun, S., & Liu, Y. (2003, October). Multi-robot SLAM with sparse extended information filters. In 11th International Symposium of Robotics Research, Siena, Italy (vol. 15, pp. 254–266).

USC Robotics. (2009). The SeaBee Autonomous Underwater Vehicle. http://ilab.usc.edu/uscr/index.html. Accessed July 15, 2009.

USGlobalSat, Inc. (2009). USGlobalSat Incorporated. http://www.usglobalsat.com/p-47-em-408-sirf-iii.aspx. Accessed July 15, 2009.

Valgren, C., & Lilienthal, A. J. (2008, May). Incremental spectral clustering and seasons: Appearance-based localization in outdoor environments. In ICRA2008, Pasadena, CA.

Via. (2009). Pico-ITX Mainboard Form Factor. http://www.via.com.tw/en/initiatives/spearhead/pico-itx. Accessed February 15, 2009.

Willow Garage. (2009). PR-2—Wiki. http://pr.willowgarage.com/wiki/PR-2. Accessed July 15, 2009.

Wind River. (2010). Wind River: RTLinuxFree. http://www.rtlinuxfree.com/. Accessed July 15, 2010.

Wu, B., & Nevatia, R. (2007). Detection and tracking of multiple, partially occluded humans by Bayesian combination of edgelet based part detectors. International Journal of Computer Vision, 75(2), 247–266.

Xenomai. (2010). Xenomai: Real-time framework for Linux. http://www.xenomai.org/index.php/Main_Page. Accessed July 15, 2010.

XTX Consortium. (2009). XTX: About XTX. http://www.xtx-standard.org/. Accessed July 15, 2009.