Low cost, high performance robot design utilizing off-the-shelf parts and the Beowulf concept, The Beobot project.

T. Nathan Mundhenk[a,d,*] , Christopher Ackerman[a,**], Daesu Chung[a], Nitin Dhavale[a], Brian Hudson[a], Reid Hirata[a], Eric Pichon[e], Zhan Shi[a], April Tsui[b], Laurent Itti[a,c,***]

[a]Computer Science Department, Henry Salvatori Computer Science Center, University of Southern California, Los Angeles, CA 90089-0781;
[b]Art Center College of Design, ID Studio, 1700 Lida Street, Pasadena, California 91103-1999;
[c]Neuroscience Program, Hedco Neuroscience Building, University of Southern California, Los Angeles, CA 90089-2520;
[d]Aerospace Integration Science Center, Aerospace Corporation, 2350 E. El Segundo Blvd, El Segundo, CA 90245-4691;
[e]School of Electrical and Computer Engineering, Georgia Institute of Technology, 777 Atlantic Drive NW, Atlanta, GA 30332-0250

ABSTRACT

Utilizing off the shelf low cost parts, we have constructed a robot that is small, light, powerful and relatively inexpensive (< $3900). The system is constructed around the Beowulf concept of linking multiple discrete computing units into a single cooperative system. The goal of this project is to demonstrate a new robotics platform with sufficient computing resources to run biologically-inspired vision algorithms in real-time. This is accomplished by connecting two dual-CPU embedded PC motherboards using fast gigabit Ethernet. The motherboards contain integrated Firewire, USB and serial connections to handle camera, servomotor, GPS and other miscellaneous inputs/outputs. Computing systems are mounted on a servomechanism-controlled off-the-shelf "Off Road" RC car. Using the high performance characteristics of the car, the robot can attain relatively high speeds outdoors. The robot is used as a test platform for biologically-inspired as well as traditional robotic algorithms, in outdoor navigation and exploration activities. Leader following using multi blob tracking and segmentation, and navigation using statistical information and decision inference from image spectral information are discussed. The design of the robot is open-source and is constructed in a manner that enhances ease of replication. This is done to facilitate construction and development of mobile robots at research institutions where large financial resources may not be readily available as well as to put robots into the hands of hobbyists and help lead to the next stage in the evolution of robotics, a home hobby robot with potential real world applications.

Keywords: Beowulf, Robot, Vision, Biology, Low Cost, Modular, Off-the-shelf

1. INTRODUCTION

The evolution of robotics seems in many ways to mirror the evolution of the computer. Today robots can be found in many businesses and practically every major research institution. However, the promise of the common robot envisioned by many prognosticators and authors to exist in the homes and lives of the average person has yet to be fully realized in the same way the personal computer has come to be as ubiquitous as the refrigerator. As such we believe that the next logical step in the evolution of robotics is to place robots in the hands of the hobbyists. Additionally, these robots must be powerful and flexible enough to spur this next step. We have thus designed a

* mundhenk@usc.edu
** christoa@usc.edu
*** itti@pollux.usc.edu
http://ilab.usc.edu, http://www.beobots.org

powerful, durable yet relatively low cost robot that relies almost exclusively on off the shelf parts. By making the design open source, it is our goal to create the next generation of robots for the home hobbyist and for research institutions without deep pockets.

It is our belief that by putting a robot in the hands of the hobbyist, more players can be brought into the game. This will allow more minds to be applied to the problems that face robotic development and the barriers to full domestic utilization. Further, as the hobbyist begins to "play" with robots, industries will have incentive to develop robotic devices for home use to meet the market of the home robot enthusiast. This will not only create new markets for robotic devices, but will create economy of scale for current robotic devices as well. However, as mentioned in order for this to happen robots must be cheap enough for a hobbyist to afford. Additionally, the assembly and usage of the robot must be easy enough such that extensive technical training is not necessary.

While the usage of off the shelf parts and an open source design principle makes the assembly of the hardware components easier, it is also necessary that software implementation be easy to understand, as well as effective enough that the hobbyist can create robotic applications that are useful in the home. This is analogous to computer programmers writing simple programs to balance their checkbook in the early days of home computing. While such applications may not have been efficient for their time, it created the foundation for idealistic development that would lead to the spreadsheet and other useful home applications that came later. As such, we are developing a comprehensive open source toolkit based upon biological principles to bring powerful software applications to the end user hobbyist to experiment with in the hopes of creating highly useful home and real world applications.

2. ROBOT DESIGN

2.1 Hardware Design

Our robot, which we have named Beobot is the product of the emerging power of open source software as well as the entry into the market of consumer grade robotic devices that previously only existed in industrial as well as scientific applications. For instance, servomotors are now widely used in remote control (RC) hobby vehicles. Given the nature of RC racing, these servos must be cheap, durable and have ample torque for their size. Additionally, the motors used to run RC cars have become more powerful allowing for the construction of larger lower cost RC vehicles. The Beobot is based upon such a vehicle. We chose the Traxxas E-Maxx RC car because it was one of the largest electric RC cars on the market. It is a 4-wheel drive model truck that is able to reach speeds over 35 MPH, and is servo controlled. It also includes many benefits such as independent shock mounted suspension, differential power control to wheels and a shift on the fly two-speed transmission. In addition to the drive motor, steering is also servo controlled. This provides an easy interface to computer control. This is achieved through the low cost Mini-SSC II servo controller, a device the size of a silver dollar, which allows interface between computer serial RS-232 interface and servomechanisms of the type used in RC cars. Thus, the task of creating a robot in the E-Maxx is a matter of connecting the servomotors to the Mini-SSC to a serial port on a computer. In essence, the minimum hardware requirement is only three components (1) E-Maxx RC car (2) Mini-SCC II (3) a computer. Additionally, it should be noted that many RC cars including gasoline powered RC cars are also servo controlled. Thus, with some creative judgment, the base component vehicle could take on many forms. At this point we should also note the limitations of using off the shelf servos; for instance, the Mini-SSC has a resolution of 256 pixels. Thus, it is not ideal for certain industrial and scientific applications where high accuracy from servos is needed.

The design of a robot is of course more than taking a computer and dropping it on a drive train. The choice of the computer is also important. The server market for computers has created an ideal computer form for our robot called PICMG. These are compact motherboards for use in small 1U size rack mountable servers. These motherboards tend to include many on board features such as video, firewire and gigabit Ethernet. We chose the Rocky-3742EVFG, which is a dual 1 GHz Pentium III based motherboard. The gigabit Ethernet allows us to connect two motherboards into a Beowulf style distributed computer. Integrated firewire creates in input for streaming digital video (30 fps) at 640 x 480 resolution. Integrated USB and serial ports connect to peripherals such as GPS, WiFi Ethernet, Mini-SSC and LCD readout panel. Additionally, the motherboard has a built in slot for compact flash. This allows us to install mission critical software and boot components onto a flash disk that is more resilient to shock than a standard hard drive. Figure 1 shows the Beobot opened for view with a basic schematic.

Power is provided from Lithium Ion or Nickel Metal Hydride based batteries. The power source for the car is kept separate from the computer power source. The computer uses eight 7.2 volt power cells. These send power to one of the few custom designed components of the Beobot, which is the computer power supply. This uses the Texas Instruments Power Trends chip set to create +/- 12 volt and +/- 5 volt power supply. This powers the computer and any standard computer peripheral such as hard drives and USB devices. Power to the robot computer can be maintained for one to two hours depending on the type of power cells used. Robot drive power runs about 20 minutes. This does not present as a problem since the battery cells are small and easy to replace.

By encasing Beobot in a hard plastic shell and utilizing base components made of Lexan, our robot is extremely durable and able to withstand high-speed impacts. This is an important aspect of robot design as autonomous devices do not always obey our will. While a remote e-stop has been included via a switch from the RC receiver, human error is inevitable and it is desirable to scratch and dent our robot rather than disable it. Repeated operator error has demonstrated that our robot is more durable then we had originally anticipated. Additionally, none of the incidents of error has resulted in any disability to the robot in a temporary or permanent fashion.

Not including the cost of tools and labor, the total cost of the Beobot is around $ 3,800. This includes two dual Pentium III mother boards, eight 38 Watt Hour Li Ion batteries, the E-Maxx RC car and all peripherals mentioned.

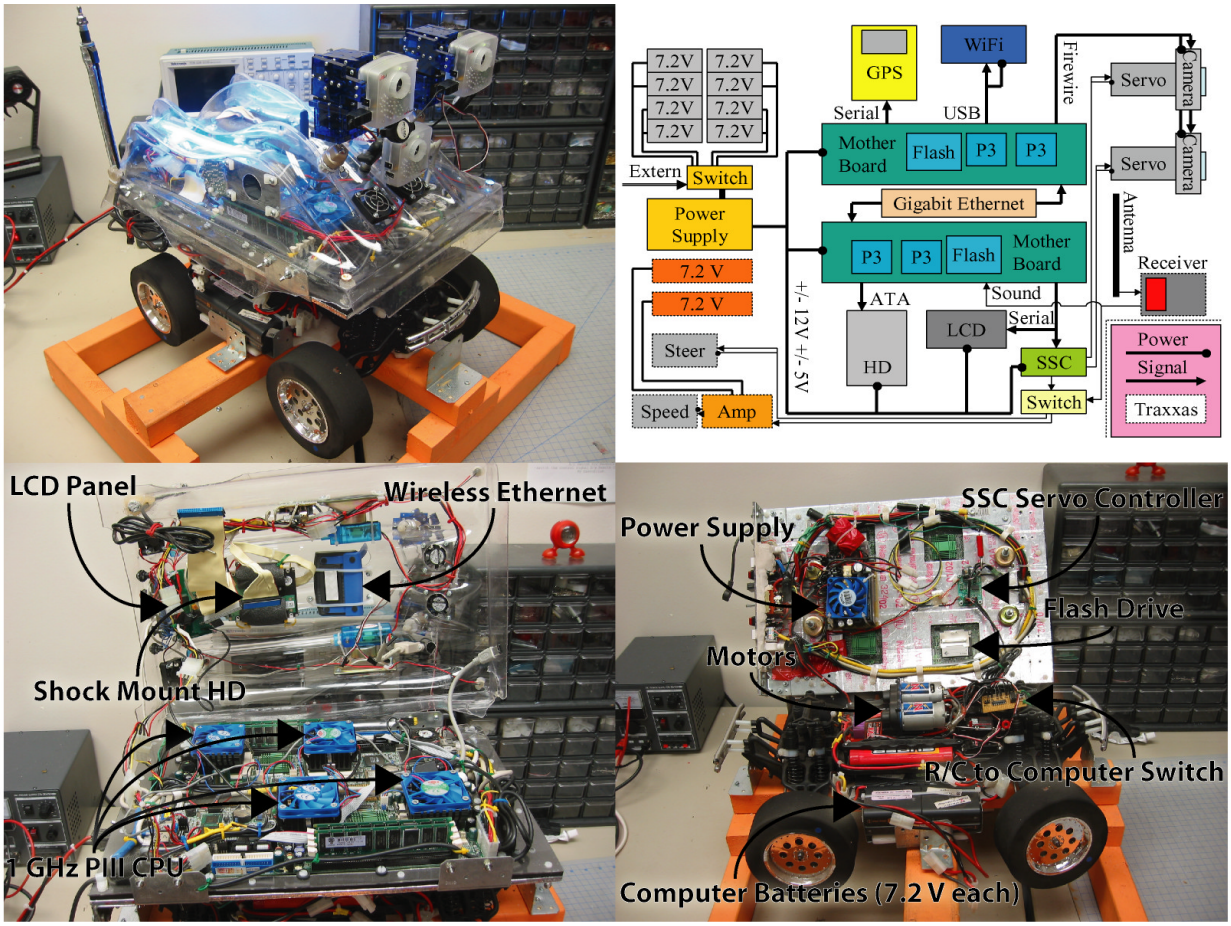

Figure 1: This is the schematic and layout for Beobot. The top left box is the completed Beobot robot. The top right box shows the linking of two power sources from batteries to primary components such as motherboards and signal outputs such as serial and USB to peripheral components. The bottom left is the robot with the top plastic shell opened. The frame in the bottom right is with the next layer opened which is the motherboard base plate.

Part	Quantity	Cost per unit	Total Cost
Rocky 3742 Motherboard	2	500	1000
512 Mb Memory (PC 133)	2	36	72
Flashram (256 Mb)	2	47	94
Unibrain Fire-I Camera	1	100	100
Notebook Hard Drive (40 GB)	2	107	214
Traxxas E-Maxx	1	369	369
1 GHz Pentium III CPU	4	92	368
SONY NP-F960 7.2 V 38.8 Wh Battery	8	129	1032
Power Supply (TI Power Trends + Parts)	1	300	100
Extra Parts, misc	1	200	500
		TOTAL (USD)	3849

Table 1: This is the breakdown of costs of building the Beobot in US Dollars. It includes all the major components with current prices.

We have also factored in $500 for extra items. Table 1 shows a break down of this. From a cost standpoint, this places the Beobot mobile robot in the price range of the average hobbyist as well as smaller research institutions.

2.2 Software Design

In order to obtain the greatest amount of flexibility Beobot is built on the Linux operating system. All code is developed and compiled with the open source GNU C++ compiler and libraries. Software interfaces to hardware as well as vision processing programs were developed as part of the open source iLab Neuromorphic Vision Toolkit (INVenT). Thus, every single software component of the Beobot is open source and freely available. Additionally, GNU open source software is generally far easier to port to other platforms than closed source software. Thus, the software components may be ported to other operating systems such as BSD or Mac OS X. This allows greater flexibility to a second party developer who may prefer a different operating system platform.

Hardware interfacing software to the Mini-SSC and LCD panel are relatively simple and involve no more than simple C++ write commands to the serial port. Inputs from video source use the INVenT frame grabber class to get video into a program usable format. Additionally, INVenT creates Beowulf interfaces for distributed computation. Manipulation and processing of images is handled from the INVenT image processing class.

3. APPLICATIONS

3.1 Leader Following

One reason for the development of Beobot is to test primarily vision based programs. As such we have limited Beobot's other senses for now. This means that a leader following algorithm must be strictly visually based. Since Beobot includes four 1-GHz CPUs this allows us more levity to create complex vision algorithms. Here, leader following is done by a more traditional blob tracking algorithm but it can also be done by biologically based saliency.

To leader follow using blob tracking we have created our own smart blob tracker. It integrates the ability to adaptively track in HSV color with multi blob tracking and segmentation. This allows the program to analyze multiple blobs and pick which ones it believes are the best candidates for tracking. Adaptive HSV color thresholding allows the tracker to adjust to variable lighting conditions and changes in intensity. The adaptive nature is also

essential since many consumer video sources have their own light adaptation, which can change the perceived hue and saturation of the target on the fly. Figure 2 gives a visual example of the outputs from the tracker.
The blob finding and segmentation algorithm proceeds in several steps.

(1) Find HSV color candidate pixels
- Scan the image and do an initial thresholding on pixels based upon some desired HSV threshold.

(2) Remove singles
- Remove candidate pixels without at least one 4 connectivity neighbor since it is most likely noise.

(3) Scan image and Link pixels into blobs
- Scan the image and link candidate pixels into discrete blobs. Link connected blobs into unified blobs.

(4) Determine blob viability
- Based upon Mass, dimensions, Ratios and Trajectories with hard constraints or soft statistical constraints, remove blobs less likely to be the target.

The first step involves labeling pixels as candidates if their Hue, Saturation and Luminance values are within a boundary either set by the operator or maintained by the adaptive HSV thresholder. For the adaptive thresholder, HSV bounds are based upon mean values from past frames for the target blob with the threshold limit set by the standard deviation. All candidate pixels are marked with a bool value that indicates that they are a candidate. Pixels that are not candidates following this step are ignored. Some noise pixels are eliminated by removing singles. That is, if a pixel does not have a 4 connectivity neighbor, it is removed from the list of candidates.

To link blobs, maps and tables are used to label a pixels membership in a blob. This increases memory used, but decreases CPU overhead. Figure 3 shows an example of this algorithm at work. The end result is several blobs that

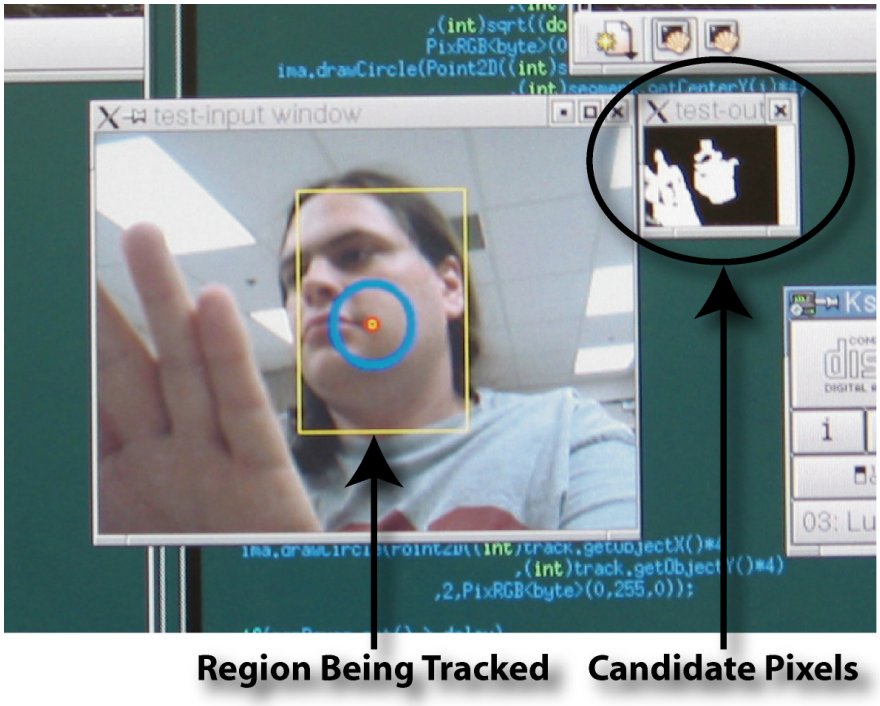

Figure 2: In this example the tracker is keying off of skin color along with the basic size ratios of the human head. Here it is quite adept at finding and tracking the human head even in the presence of a large number of distracters. Here is can be seen that the tracker is not fooled by the introduction of even the experimenters hand into the scene. This can be observed even though, in the candidate pixels window, the hand as a larger correct color target. Since the hand has slightly different color characteristics, even upon path crossing, the program still tends to follow the head due to adaptive color thresholding.

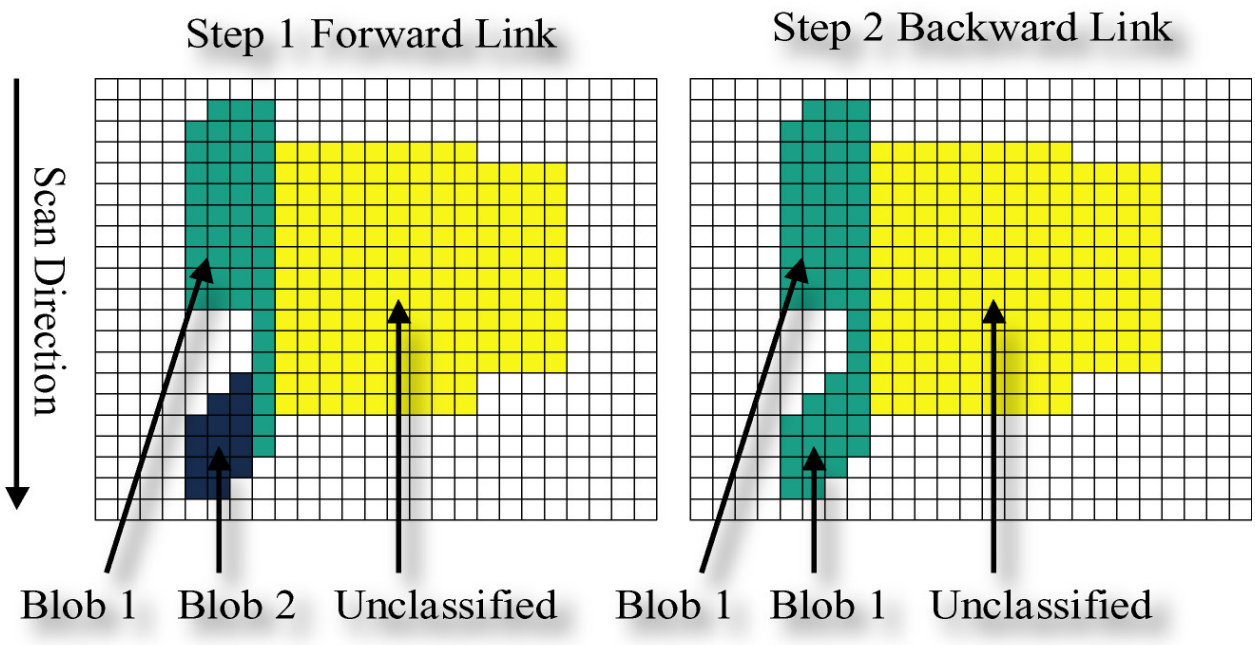

Figure 3: The algorithm must find blobs without knowing how many there are. It first finds all candidate pixels by applying HSV thresholding. All candidates are linked together during the image scan. This is done by labeling each contiguous scan line. If two contiguous scan lines touch, they are labeled as being a member of the same blob. As can be seen in step one, this might lead to the same blob being labeled as two distinct blobs. To avoid this, as seen in step 2, scan lines check to make sure their neighbors belong to the same blob, if not, then scan lines are backward linked into the same blob. This requires that maps are created that show how pixels have membership in contiguous scan lines which have membership in blobs. Maintenance of these maps also facilitates post hoc statistics on blobs since each blob is treated as an independent entity. The down side for this method is the memory needed for the maps, but this is not excessive and given that RAM is cheap, a reasonable strategy.

are distinguished as being discrete if they do not touch. The resulting blobs are then processed for fitness. If a blob is too large or too small it is eliminated. This can be determined by both pixel count mass and the size of the bounding box which encloses it. The blob can also be eliminated if its proportions are incorrect for the target. Using statistical methods, a blob can be eliminated if it violates expectations for size, trajectory or position.

The remaining blobs can be treated in a number of different ways. For instance, a blob that is most like the blob in the last video frame could be voted as a winner blob. However, we have had success with simply combining the remaining blobs into a mother blob and tracking that one. This is less computationally expensive and seems to yield good results. The computational epilog for the blob tracker is that it processes and tracks multiple blobs in real time (30 fps) using about 25% CPU on an AMD 2000 XP based system. This figure includes CPU overhead from several video stream outputs to X Windows.

Beobot uses the tracker to establish the position of the target relative to itself. Steering is adjusted proportionally to keep the target collinear with its trajectory. Speed is adjusted based upon an observed homeostasis in blob target size. The robot attempts to keep the target size constant by slowing down or speeding up. Since uncertainty is constant in such a simple paradigm, beobot must use cautious heuristics in speed adjustment. For instance, assuming the target is moving fluidly, size should not change suddenly. If the blob changes suddenly the robot attempts to smooth over such events, for instance, with a Kalman filter. This prevents sudden surges of speed, which may be hazardous to both operator and robot.

3.2 Visual Robot Navigation Using Image Spectral Information

One of our interests is in exploring what navigationally useful information can be extracted from the spatial frequency and orientation components of an image, and to apply this information to the task of autonomous robot navigation, in this case path following.

For this job we do not attempt any computationally expensive and time-consuming tasks such as object recognition, so as to enable us to operate in real time, at relatively high speeds (brisk walking to jogging, limited primarily by safety concerns). This also prevents us from being dependent on specific environmental features and creates a more biological-like flexibility.

For this we use the Fourier amplitude spectrum, which reflects an image's dominant contours, their orientations, widths, and lengths. This information should be able to describe the relevant features of a path we wish the robot to follow. It also reflects the degree of high- versus low-frequency information in an image, and combined with the above can describe the potential for movement within a scene.

To demonstrate that high-level, human meaningful navigational descriptors can be extracted from the spectral components, we first classified images along the following dimensions: path vs. non-path, orientation/direction of path, depth of scene, and obstacles in scene.

That there is sufficient information in the amplitude spectra to make the discriminations described above is supported by the averaged Fourier amplitude spectra of iconic images of each feature extreme, as illustrated in Figure 4. Here we have pictures, which were taken on the University of Southern California (USC) campus at slightly above ground level, at a robot's eye view, and thus are dominated by path borders, if present, and other potential ground-level obstacles or barriers.

Ideal paths are generally indicated by roughly parallel lines, which indicate path borders, extending off into the

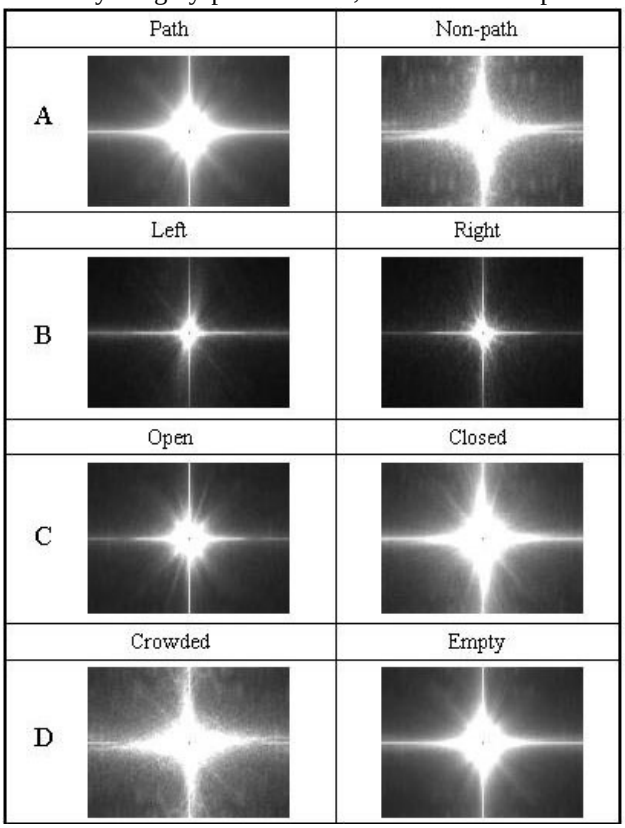

Figure 4: Averaged Fourier amplitude spectrum for all images human-labeled as path or non-path (A) and highly leftward- or rightward-oriented (B), open (deep) or closed (C), and crowded (with obstacles) or empty (D).

distance. As shown in Fig. 4A, this is reflected in the averaged amplitude spectrum of path scenes by higher activity in the diagonal orientations compared with the vertical and horizontal ones, unlike the non-path scenes, which are more dominated by vertical and horizontal orientations, and which also have more high-frequency content, reflecting the relative smoothness of the path portion of path scenes versus non-path scenes.

The information critical to staying on a path is the direction along which the path is oriented, and this information is also reflected in the Fourier amplitude spectra. As can be seen in Fig. 4B leftward-oriented scenes have increased activity in the first quadrant (i.e., the path is at ~135 degrees, meaning the borders are oriented so that they appear in the intensity gradient as a wave moving at ~45 degrees). The rightward-oriented scenes show the complementary pattern.

Open scenes, with high depth, have less high-frequency energy than closed scenes, whose close-up obstacles presumably supply high-frequency details not visible on the more distant barriers in the open scenes (Fig. 4C). Finally, scenes crowded with obstacles contain more high-frequency energy than empty scenes, for reasons similar to above, and also are somewhat biased in the horizontal frequency direction, corresponding to vertical lines in image space, which perhaps reflect vertical obstacles such as people (Fig. 4D).

We have compiled a data set comprised 1,343 RGB images taken from a Beobot-mounted camera. Some of these were from a consumer-grade Hi-8 video camera (Sony, Inc.) and some from a firewire consumer web camera (Unibrain Fire-i), which is the Beobot's normal means of vision. Images were manually labeled along the dimensions of left-right orientation, depth, and obstacles, as well as whether they were path, non-path, or ambiguous. For orientation, images were put into one of eight different classes. For depth and obstacles, images

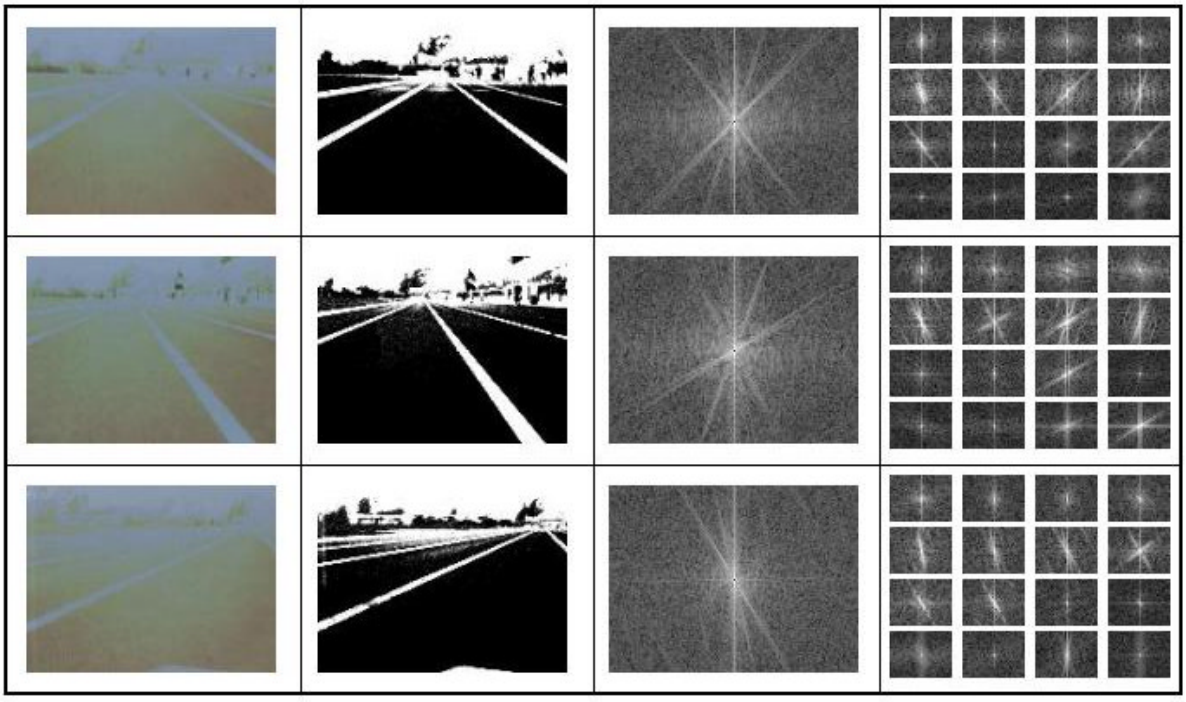

Figure 5: From left to right, the original image, the brightness-only and normalized image, the amplitude spectrum from the Fourier transform of the image, and the amplitude spectra from the Fourier transform of each of 16 nonoverlapping 30x40-pixel blocks of the image (for 3 Beobot-captured track images, showing straight, leftward turning, and rightward turning instances, from top to bottom).

were put into one of six different classes. Of these, 896 images were used for training, 447 for testing. The images

were all pictures of daytime outdoor scenes taken on the USC campus, especially but not exclusively along paths, and along the school's running track.

It is important to note that nothing in our technique has been specifically designed or tuned to the class of images used here for testing. Presumably, similar results could be obtained in very different environments, such as in indoor situations. Pictures were either 120x160 pixels or were 240x320 pixels; the latter were downscaled to 120x160. Pictures varied not only by location and resolution but also by time of day, and thus overall illumination, and by camera angle.

RGB images were converted to HSV, with only the value component being retained. Each image was normalized for luminance by subtracting out its mean and dividing by its standard deviation. Next, each image was discrete Fourier transformed. The results of these processes can be seen in Figure 5.

Because we are only interested in non-localized frequency information, we discard the phase information and use the amplitude spectrum. The amplitude spectrum is a 120x(160/2+1) = 9720-element array of real numbers. In order to reduce dimensionality sufficiently to make the learning tractable for the neural network, we convolve the image with 40 filters of varying scales and orientations. The filters are log Gabors tuned to five spatial frequencies, with eight different orientations at each frequency. The dot products are computed and the resulting 40 scalars are saved as a feature vector.

Feature vectors for the 896 training images were used to train a fully connected feedforward neural network using backpropagation. For online use, test images were processed as above and then passed through the feedforward network to get an estimation of the four high-level features.

This classification results for the global amplitude spectrum, completely nonlocalized within the image are seen in Table 2.

Condition	Nonlocalized classification		Coarsely localized classification	
	Accuracy	Mean Squared Error	Accuracy	Mean Squared Error
Path vs. Non-path	81%	0.22	84%	0.18
Orientation	40%	1.56	58%	1.23
Depth	60%	0.52	56%	0.59
Obstacles	48%	1.01	57%	0.68

Table 2: Accuracy in judgment for different kinds of conditions using non-localized and localized classification

To see if coarsely localized information would help the classification, we next performed the above procedures not on the entire image, but separately on 16 non-overlapping 30x40-pixel blocks. This leaves 16x40=640 features. To reduce this to a more manageable number for training, we took the first 40 principal components of the features computed over the training set, yielding the results on the test set seen in table 2.

Several different network architectures, as well as several simpler linear classifiers all yielded similarly superior results for the coarsely localized model over the global one. There are significant gains in orientation discrimination and obstacle detection. We therefore used this model for navigation.

Having demonstrated that navigationally useful information can indeed be extracted from image spectral components, we next developed a system to use this information for autonomous robot navigation. Orientation judgments can be mapped straightforwardly onto steering commands. To enable continuous improvement through online training, we use a neural network that maps the processed visual features directly onto steering commands and learns from its errors in real time. The current architecture is a 3-layer backpropagation network with 40 inputs and 13 outputs, ranging from hard left to hard right, probabilistically chosen at each iteration based on strength of response.

Initial weights may be learned offline from images and corresponding steering commands previously captured with the robot under human control, or they may be set randomly.

Then, the operator takes the robot out to the chosen course and lets the robot run autonomously, using the autonomous steering commands that the algorithm computes. As the robot runs along, the operator can correct the steering choices as necessary via remote control. These human-issued commands are treated as target values from which an error is computed for backpropagation. The path-following performance of the robot thus improves with time.

4. DISCUSSION

4.1 Basic Hardware and Software Issues

The amount of CPU power provided by the Beowulf computer in Beobot is more than adequate for even complex visual tasks. Additionally, its price tag is low enough to give smaller institutions and hobbyists a chance to experiment in the field of mobile robotics. However, several practical limitations exist with the current robot. The first is the battery limitation. While the batteries for the Beobot are more than adequate to drive it during experiments, since they are separate cells, recharging can be time consuming. A solution to this is to purchase more battery chargers from 50$ to 150$ each depending on the type of cells you are charging and the bells and whistles involved. This can increase the cost of maintaining the robot substantially if you build a one to one population of chargers for each battery. The alternative is to charge each of the 10 batteries the robot uses separately in order. That is, if you opt for two chargers you must charge five sets of two. The toll is that a person needs to be around to supervise the recharging. This illustrates by far the largest single barrier to keeping Beobot ready for play. However, even with the problem posed by powering the Beobot with off the shelf batteries as we have done, the Beobot is still able to run every day for experimentation.

Another important issue to mention with the Beobot is that while the Linux operating system environment is well suited for development, it is sometimes limited in its hardware support. For most important devices, drivers and support utilities have been written. However, there are many situations in which drivers have not been written for hardware you may wish to install. Thus, there is some limit placed on how easily new hardware may be integrated onto the Beobot. This limit is however tempered by the fact that creating drivers on an open source platform is generally easier than for a closed source OS since the operating system mechanisms are completely visible. Additionally it should also be noted that the open source concept is pivotal to development in a Beowulf cluster since the Beowulf networking mechanisms are not only more developed under Linux, but require open sourcing of networking services in order to customize them for the distributed computational task at hand. It is also important to note the power of customization available with Linux and other open source operating systems. This allows Beobot's operating system and utilities to be squeezed onto a 256 Mb flash disk along with the test software. Further, we can avoid loading a windows operating environment and gain efficiencies from not having the computational overhead that it comes with.

One of the strongest characteristics we have observed with Beobot is its durability. Since it is built on top of an RC chassis built for "Off Road" racing, it has the ability to take a great deal of stress. Additionally by supporting other Beobot components with lightweight highly durable materials such as Lexan, Beobot is able to withstand strong impacts and generally rough treatment. Even if parts of the robot break in response to collision or other trauma, most components that make up the exterior and supporting frame are low cost and easily replaced. This creates a certain liberty during experimentation since there is less worry that the robot will damage itself in the event it misbehaves.

4.2 Operational Issues

One major issue yet to be resolved is how to create the most ideal interface for Beobot to use during experimentation. For instance, our current interface uses a two character line LCD panel with five input keys to run the robot during experimentation. This is a low cost approach, but it limits the interaction between the operator and robot during trial runs. To address this issue, future plans may involve a wireless connection via a handheld

computer. This would allow us to use a portable graphical interface with greater flexibility to control and interact with the Beobot than a terminal that is hard mounted onto the robot. With the advent of high-speed wireless access at speeds of 56 MB/s, streaming important data to a handheld computer is highly viable.

4.3 Software Development issues

Utilization of INVenT facilitates development and deployment of robotic applications. With software modules for control, network communications and image manipulation it becomes easier to create applications for the Beobot. For instance, a model manager handles each software function of the robot, which is a step towards creating a Lego like environment for robot development. Software components for controlling motion, creating network communication, grabbing input and processing information can be reused and are portable to other robotic applications. For instance, the leader following and track following programs reuse the same code for distribution of processes across the Beowulf cluster. They also use the same methods to grab images and manipulate them. Thus, in order to create and deploy a new application for Beobot, the programmer can skip development of many low-level systems. For instance, the code for leader following was created and deployed in the equivalent of a few weeks rather than months by using this readily available code base. Additionally, by creating the leader following code in a modular fashion, it too can be used as a building block onto which other robot applications can be developed.

5. CONCLUSION

The Beobot has allowed us to test a variety of both biological and traditional algorithms in the real world with success. Given its relatively low cost and high performance coupled with its off-the-shelf modular design should enable other labs and individuals to borrow from our design and create their own low cost robots. This will hopefully enable powerful robotics to come into the hands of the hobbyist and labs with smaller budgets, thus accelerating the development of robotics into the home and other areas where robot deployment is still sparse. From our experience with Beobot, it should be replicable by other individuals and facilitate the next stage in the evolution of robotics, creating a home hobby robot with enough computational power to facilitate the development of useful home applications.

ACKNOWLEDGEMENTS

We would like to thank and acknowledge the Beobot Team members past and present for the contributions. I would also like to thank Mike Olson for his help and suggestions. This research is supported by the National Imagery and Mapping Agency, the National Science Foundation, the National Eye Institute, the Zumberge Faculty Innovation Research Fund and the Charles Lee Powell Foundation.