AutoPower: Toward Energy-Aware Software Systems for Distributed Mobile Robots

Keith J. O'Hara, Ripal Nathuji, Himanshu Raj, Karsten Schwan, Tucker Balch
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332–0250
Email: {kjohara, rnathuji, rhim, schwan, tucker}@cc.gatech.edu

***Abstract*— Autonomous robot systems have to manage their energy wisely in order to complete their missions. Typical approaches seek to conserve energy by energy-efficient motion or sensor planning. This paper puts forth a distributed systems approach to power management. Specifically, it develops and presents AutoPower, which is a model that characterizes robot software systems' computation and communication energy behaviors. With AutoPower, it is possible to make principled decisions about (1) where to deploy software components across the distributed computing resources of autonomous robotic systems, and (2) how the different systems involved should communicate to best meet overall mission objectives. We showcase AutoPower by using a multi-robot search-and-rescue mission as a guiding application. For this scenario, application of the model shows that there are counterintuitive energy trade-offs in configuring such application software. Further, by using AutoPower to guide deployment and interconnects at runtime, for certain configurations, overall computing system lifetimes can be increased by up to 57% over a base-line configuration.**

I. Introduction

Autonomous robot systems have to manage their energy wisely in order to complete their missions. Typical approaches seek to limit the energy usage of each robotics system platform, using general or application-specific power management methods. Examples of the latter are energy-efficient motion or sensor planning. We seek to avoid solutions that limit individual robots' sensing or actuation capabilities in order to ensure longer lifetimes for their computing subsystems. Toward this end, this paper presents a distributed systems approach to runtime power management. Specifically, it develops and presents the AutoPower model for characterizing and manipulating the software systems that execute on robotic platforms. The AutoPower approach:

- offers a general method for modeling robot software systems' computation and communication energy behavior, and
- by using this method, principled decisions can be made about (1) where to deploy certain software components and (2) how communications between components should be realized.

There are multiple reasons why AutoPower adjusts robot software systems' energy behavior without considering trade-offs that may be realized by changing robots' sensing or motion behaviors. First, solutions like AutoPower can support efficient energy usage without requiring roboticists to change the behaviors they are seeking to develop or utilize. AutoPower's energy management solutions, therefore, are a general service to any robot system regardless of its particular application, architecture, robot platform, etc. Additional energy savings attained via application-specific solutions will simply enhance AutoPower-based solutions. Second, as autonomy increases in robot systems, energy usage for computation will constitute a substantial portion of total energy consumption (i.e., battery drain). Third, as we look to teams of robots, system energy consumption is increasingly impacted by communications across team members.

Decisions about which software components to run on the computer systems of distributed robots depend both on current application needs and on available system resources. Application-level decisions that affect the use of available system resources include the following:

- **Which components should run?** Concrete examples are which localization, path-planning, or vision routines to execute on robots.
- **How should the components run?** Application needs dictate how many particles to use for Monte Carlo localization or what grid granularity to employ for D*.
- **When should the components run?** Real-time constraints are determined by current application context or mission parameters, determining for instance, the periodicity and deadline for the obstacle-avoidance routine.

The AutoPower approach seeks to retain for applications the ability to run the components they desire and in the fashion suitable for their current needs. Rather than limiting the application's ability to use system resources, the approach supports application needs by seeking to automate how such resources are used:

1) **Where should the components run?** For example, should we 'offload' localization computations to a remote computer?
2) **How should the components interact?** Should robots communicate using 802.11a or should they use bluetooth communications? Should component middleware use pull- vs. push-based communications?

Stated more explicitly, AutoPower attempts to answer the following question: **How can we deploy a robot software system to maximize the lifetime of the system while**

still meeting application QoS requirements? In addition, AutoPower looks to manage energy at a system-wide level, rather than maximizing individual systems (e.g. communication protocols, operating system scheduling) of individual robots (e.g. voltage scaling, sensing frequency). This paper is a first step toward system-wide energy management for distributed autonomous robot systems.

II. The AutoPower Energy Model

Our first goal is to model a robot software system's energy behavior. The composition of such a model requires three main components: (1) energy characteristics of computational platforms, (2) energy characteristics of communication mediums, and (3) execution profiles of software components.

The first key component is the energy characteristics of the computational resources. It is well known that the precise power consumption of a workload can vary with respect to use of the processor, much less the additional consumption of resources like memory, external buses, etc. While fine grain power models attempt to capture all such usages, the AutoPower approach focuses on the scalability of power modeling, by offering a first order approximation of total system power usage. The goal is to capture key system-level tradeoffs. AutoPower attains this goal by summarizing the energy costs of computational resources in terms of joules per millisecond of computation. This allows us to describe the energy demands of our software simply by latency measurements, rather than direct energy measurements, which are more time consuming and costly to gather.

In this work the computational model is developed based upon real hardware. Specifically, the Intel Sitsang experimental platform is used as a typical on-board computational platform for robots. The Sitsang is based on the Intel PXA255 XScale processor and is aimed at embedded platforms like PDAs and robots. This platform would be ideal for applications such as robots since the PXA255 provides integrated solutions for peripherals such as flash, memory, infrared, USB, and others. Moreover, the solution strikes a solid balance between performance and power. Though the XScale core lacks a floating point unit, it can be clocked up to 400 MHz, and is accompanied with 64 MB of RAM, 32 MB of Flash memory, and supports dynamic voltage and frequency scaling. The platform itself is capable of various wireless communications such as bluetooth and 802.11a via expandable slots.

To develop our computational energy model, various workloads are executed on the Sitsang platform, and platform power measurements are performed using a Tektronix TDS5104B oscilloscope, Tektronix TCP202 current probes, and Tektronix P6139A voltage probes. It should be noted that though the Sitsang is equipped with a LCD display, since these might not be attached on a deployed robot, the LCD, backlight, and framebuffer DMA were disabled for our measurements. The workloads utilized were various media based encoders/decoders that are typical of systems such as robotics. When running at 400Mhz, the average computational overhead when idle is 1.34W, and 2.27W when active.

	802.11a	Bluetooth	GSM
Bandwidth (kbps)	54000	1500	116
Sleep Power (W)	0.010	0.000036	0.000165
Send Power (W)	1.498	0.095	1.815
Send μjoules/bit	0.028	0.064	15.647
Receive Power (W)	1.22	0.095	0.165
Receive μjoules/bit	0.022	0.064	1.422

Fig. 1. Energy characteristics of communication resources.

The second component of the system energy model involves describing the communication infrastructure. The communication resources are characterized by the average amount of energy (joules) needed to communicate one bit of information. 802.11a, Bluetooth, and GSM (mobile phone network) are considered as possible communication mediums. The specific energy characteristics of each radio are taken from [7] and are reproduced in Figure 1.

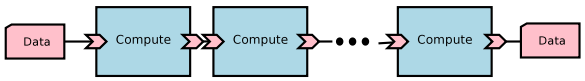

Fig. 2. Application Chain: Our application execution model.

The final piece of the energy model is concerned with the software components that must actually execute in the system. In order to derive energy costs, software components are modeled as chains of computation and communication. Each node in the chain does some amount of computation and passes along data that serves as input to the next computational node. The nodes are described in the applications chains by their average computational latencies, and the links of the application chains are described by the sizes of data communicated between two adjacent nodes. The model is illustrated in Figure 2. In order to accommodate application-level requirements (formulated as quality of service (QoS) constraints) into the model, each chain must specify its timeliness requirements (e.g. rate) as well as any hardware constraints like the need for the presence of certain sensor devices or human operators. The overall modeling process is visualized in Figure 3.

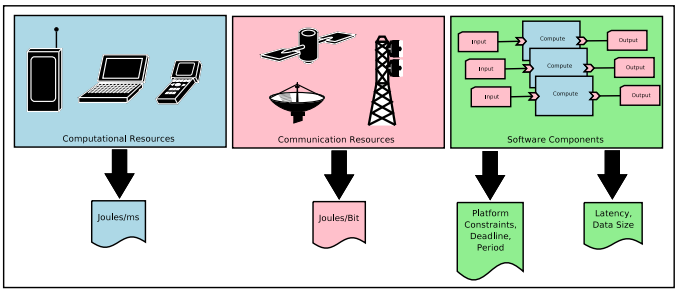

Fig. 3. The process of building an energy model.

III. Application Scenario

In order to demonstrate our energy modeling approach, a search-and-rescue mission is used as a guiding application

throughout this paper. Specifically, the scenario considered is where a multi-robot team is tasked with searching a building for victims. The lifetime of the system is defined as the amount of time until one robot depletes its energy reserves.

The robot team consists of three mobile robots, all with laser-scanners and odometers. Two of the three robots are equipped with cameras. The third robot acts primarily as a communication relay. The robots communicate to a base-station (a laptop) operated by a human operator. Also, the robots can use the base-station computer for off-loading computation if necessary. The base-station can communicate with the robots using either GSM or 802.11a (if range permits) communication. The robots have embedded Linux computers on-board and can communicate to each other using 802.11a or bluetooth (again, if range permits). Lastly, the computers are also able to route messages between each other using a shortest-path routing algorithm.

We focus on two software components that will be executed in support of this mission. First, a piece of vision software that detects possible victims. Our particular implementation (blob-finder) locates blobs of different colors in an image. The information from the blob-finder is communicated to the human operator for victim/non-victim classification. The application demands the blob-finder run 4 times a second.

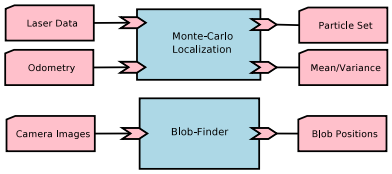

Fig. 4. Two application chains for the search-and-rescue mission.

In addition, the robots use Monte-Carlo Localization to estimate their position in the building. The robots may need to report their positions to the human operator, but not necessarily continually. The localization components should execute once every two seconds. The two application chains for these components, as described above, are illustrated in Figure 4.

IV. Two Software Execution Profiles

Given the application described in the previous section, energy profiles are created of Monte-Carlo Localization (CARMEN [11]) and Blob Finding (CMVision [1]). While relevant to our driving application, these applications are also representative of typical robotics software. First, both implementations are taken from popular open-source robot software distributions. Second, the two pieces of software have very different computational behaviors, which makes them an interesting pair to energy-profile. Specifically, MCL makes heavy use of floating point operations, whereas blob-finding primarily uses integer operations. This distinction causes them to differ substantially with respect to their characterization by the AutoPower energy model. The Sitsang platform lacks a floating point unit resulting in a bigger performance gap between the laptop and Sitsang when running localization compared to running the blob finder.

	Localization	Blob-Finder
Period (sec)	1.0	.25
Laptop Latency (sec)	0.0035	0.01048
Sitsang Latency (sec)	0.85	0.115
Input Size (bytes)	1059	230415
Output Size (bytes)	805	200

Fig. 5. Energy profiles for the two software components.

The average latencies and data sizes of the two applications are profiled on the computational platforms (Intel laptop, Sitsang) and recorded. The localization module is run using a simulation of two different environments. The robot in the simulation navigates from a start location to a goal position 10 times. The average localization latency is noted. The blob-finder is run on three different data-sets collected using a Logitech web-cam. The blob-finder runs on each data-set 50 times, and the average latency for processing a frame was collected. The results are shown in Figure 5. Note, the large disparity in computational performance between the SitSang and the laptop.

V. Methodology and Results

A. System Lifetime Analysis

As an initial step towards analyzing system-level tradeoffs with robotic software systems, we built a tool to compute the expected lifetime of our search-and-rescue team given our energy models, the initial energy of each robot, a description of the network (including link types and connectivity), and a particular allocation scheme. In order to limit the design space, a shortest path routing scheme is assumed for all of our network topologies.

In order to calculate system lifetime, first the base power cost of each robot's on-board computer is determined. This is done by taking the base idle power consumption (1.34W), and including the sleep power of every radio that is needed for network links. Then the application chains are mapped across the system as specified by the allocation scheme. The period information of an application chain is used along with the radio characterizations to assess the communication costs to robots. Similarly, periodicity and computation time provide computational energy costs.

To validate that all application requirements are met, the utilizations of the radio and computers are computed. If any of these values exceed 100%, the allocation scheme is deemed impossible. Otherwise, "lifetime" of the system is calculated as a function of the maximum average power consumption of all the robots in the system.

B. Static Allocation Results

Using the lifetime analysis tool, given the search-and-rescue mission previously described, the space of possible allocations and networks is exhaustively explored. Specifically, all three robots perform localization (`Loc`), and Robots 2 and 3 have cameras attached and require blob finding (`Blob`) on images.

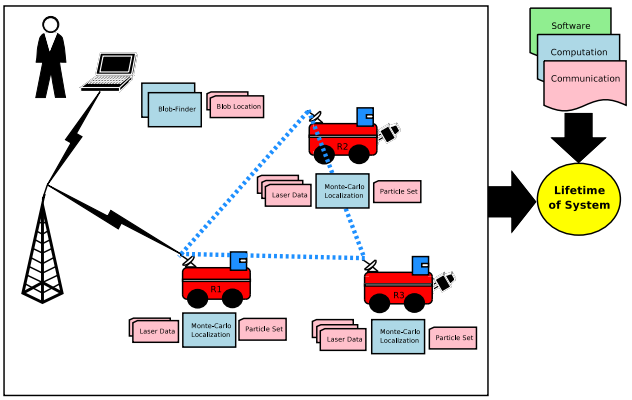

Fig. 6. An example scenario.

One particular scenario is illustrated in Figure 6. In this instance, the two robots with cameras have off-loaded their blob-finding component to the base-station. The robots are fully connected with bluetooth, and the relay-robot is communicating to the base-station using 802.11a communications. In this scenario the relay-robot routes messages between the base-station and the robots with the cameras.

In Figure 7, the results for some interesting configurations are reported. The table in the figure provides the lifetimes for various network configurations and component allocations. The "baseline" configuration is when all components execute locally and all network links use 802.11a-based communications.

The maximum lifetime configuration occurs when the network is fully connected with 802.11a and the robots offload all computation to the laptop. The lifetime of this configuration is approximately 57% better than the case where the robots perform all computations locally on board. This is an interesting result because, from Figure 1, it would seem that for power reduction, the robots should use the bluetooth radio which consumes less power. However, utilizing the more efficient radio prevents certain offloading, and therefore has a negative impact on the entire system. This type of counterintuitive tradeoff highlights the importance of the system-level deployment mechanisms offered by the AutoPower approach.

Also interesting are situations where it is unrealistic that the entire team is connected by 802.11, for instance, due to range constraints. Consider the case where the relay robot is connected to the base-station by GSM and the robots are connected by bluetooth. The relay robot's communication link is the bottle-neck in this situation. The expected lifetime of the base-line (i.e., everything executed locally) configuration is 64% of the maximum lifetime configuration. We also observe that the most efficient allocation for this network configuration is to have the robots offload their localization to the laptop. This allocation comes closer to the optimal expected lifetime, 78% of the maximum expected lifetime.

Finally, we investigate a scenario in which robots begin the mission with uneven energy reserves. In this situation, the relay robot (Robot-1) has half the energy of the other two robots. Again, interesting configurations are tabulated in Figure 8. In this scenario, the optimal configurations are less obvious. When the system is fully connected, the optimal configuration is for the robots to communicate with the base-station using GSM, and for the robots to communicate with each other using bluetooth. The robots "shift" their localization responsibilities away from the robot with the least energy. When Robot-1 is the only robot connected to the base-station the optimal configuration is also interesting. Robot-3 runs both blob-finders and Robot-2 runs the localization for itself and Robot-3. Robot-1's localization is off-loaded to the base-station.

C. The Need for Dynamic Allocation Support

Thus far, our results have illustrated the benefits of utilizing an optimal allocation scheme versus a basic local execution policy for robotic software systems. To realistically utilize these software offloading schemes, however, there must exist system support for dynamic allocation. To support this statement, consider two types of changes that may trigger a dynamic re-allocation. The first type is infrastructural changes like those due to changes in network connectivity or link characteristics. The second type is due to changes in application behavior. That is, the needs of an application, or the chain descriptions in our model, may change as scenarios unfold.

Revisiting our optimal scenario, where all nodes are interconnected by 802.11a and all computations are offloaded, let us assume that Robot-3 experiences a fault with its hardware, and must utilize GSM to contact the laptop, and bluetooth for inter-robot communications. Due to utilization constraints alone, the existing allocation scheme cannot continue while providing the required QoS. Reverting back to a local computation scheme provides only 80% of the lifetime compared to the optimal deployment, where `Loc-1`, `Loc-2`, and `Blob-2` are performed by the laptop, `Loc-3` is offloaded onto Robot-1, and `Blob-3` is performed locally on Robot-3.

Next, consider application triggered re-allocations. For instance, once the mission is over, i.e., all the vicitims were succesfully rescued, the robots may not need to use their cameras any more. In other words, during the robots' return only localization may need to be run. Previously, when the relay robot is providing the communication link back to the base-station, the configuration using 802.11A is the optimal configuration concerning total expected lifetime because 802.11 has to be used to off-load the blob-finders. When blob-finders no longer need to run, there is more flexibilty and Bluetooth is able to support off-loading the localization computations.

Finally, with support for dynamic re-allocation, the lifetime of the team could be extended even when there are no infrastructural or application changes. Previously we considered the best possible static allocation. But, by considering re-allocations, it is possible to extend the lifetime of the team even more by dynamically moving components. In other words, at a given decision point, an important system characteristic is the relative energy capacity of each robot. Given an

Lifetime (sec)	Normalized	Loc-1	Loc-2	Loc-3	Blob-2	Blob-3	Laptop-Robot Net	Robot-Robot Net
FULLY CONNECTED								
45715.2	**1.57**	**laptop**	**laptop**	**laptop**	**laptop**	**laptop**	**802.11a**	**802.11a**
29451.2	1.01	r1	r2	r3	r2	r3	802.11a	802.11a
32976.0	1.13	r1	laptop	laptop	r2	r3	GSM	802.11a
29121.6	1	r1	r2	r3	r2	r3	GSM	802.11a
45714.4	1.57	laptop	laptop	laptop	laptop	laptop	802.11a	Bluetooth
29450.4	1.01	r1	r2	r3	r2	r3	802.11a	Bluetooth
33146.4	1.14	r1	laptop	laptop	r2	r3	GSM	Bluetooth
29254.4	1.00	r1	r2	r3	r2	r3	GSM	Bluetooth
ONE RELAY ROBOT								
41820.0	**1.44**	**laptop**	**laptop**	**laptop**	**laptop**	**laptop**	**802.11a**	**802.11a**
29451.2	1.01	r1	r2	r3	r2	r3	802.11a	802.11a
35262.4	1.21	laptop	laptop	laptop	r2	r3	GSM	802.11a
29451.2	1.01	r1	r2	r3	r2	r3	GSM	802.11a
36181.6	1.24	r1	laptop	laptop	r2	r3	802.11a	Bluetooth
29586.4	1.02	r1	r2	r3	r2	r3	802.11a	Bluetooth
35432.0	1.22	laptop	laptop	laptop	r2	r3	GSM	Bluetooth
29586.4	1.02	r1	r2	r3	r2	r3	GSM	Bluetooth

Fig. 7. Expected lifetimes when all robots start with 64,000 joules of energy.

Lifetime (sec)	Normalized	Loc-1	Loc-2	Loc-3	Blob-2	Blob-3	Laptop-Robot Net	Robot-Robot Net
FULLY CONNECTED								
23697.6	1.33	r2	r2	r3	r3	laptop	802.11a	802.11a
18335.2	1.03	r1	r2	r3	r2	r3	802.11a	802.11a
23694.4	1.33	r2	r2	laptop	r3	r3	GSM	802.11a
18333.6	1.03	r1	r2	r3	r2	r3	GSM	802.11a
23696.8	1.33	laptop	r2	r2	laptop	r3	802.11a	Bluetooth
18335.2	1.03	r1	r2	r3	r2	r3	802.11a	Bluetooth
23860.8	**1.34**	**r2**	**r3**	**laptop**	**r2**	**r3**	**GSM**	**Bluetooth**
18439.2	1.03	r1	r2	r3	r2	r3	GSM	Bluetooth
ONE RELAY ROBOT								
23694.4	**1.33**	**laptop**	**r2**	**r2**	**r3**	**r3**	**802.11a**	**802.11a**
18333.6	1.03	r1	r2	r3	r2	r3	802.11a	802.11a
20812.8	1.17	laptop	r2	r2	r3	r3	GSM	802.11a
17833.6	1.00	r1	r2	r3	r2	r3	GSM	802.11a
23692.0	1.33	laptop	r2	r3	r2	r3	802.11a	Bluetooth
18332.0	1.03	r1	r2	r3	r2	r3	802.11a	Bluetooth
20947.2	1.17	laptop	r2	r3	r2	r3	GSM	Bluetooth
17932.0	1.01	r1	r2	r3	r2	r3	GSM	Bluetooth

Fig. 8. Expected lifetimes when robot-1 starts with 32,000 joules and the other two robots start with 64,000 joules of energy.

allocation that does not consume energy evenly across nodes, then, it is possible that even without changes to application chains or infrastructure, a run-time re-allocation may further improve lifetime beyond a single deployment strategy. The ability to extend optimal lifetimes with this type of reallocation will depend upon the efficiency and architecture of the system-level support for dynamic re-allocation. The next steps in our research will quantify the overheads associated with specific reallocations and then use these overheads to craft appropriate dynamic reallocation policies.

VI. RELATED WORK

Energy management is a fundamental issue in autonomous robotics. The first autonomous robots, Grey Walter's "Elmer" and "Elsie" [16], had behaviors to search for their recharging station. In much the same way the authors in [14] develop a method for a self-recharging robot to support long-lived operation. The authors of [9] present a power model of a robot based on its mechanics in order to derive the energy costs of different mobile robot coverage patterns. Our work differs from this past work because we are not putting any constraints on the behavior of the robots for energy reasons. Currently, we are only concerned with managing the energy used by the software components. Similarly, the authors in [4] compare the energy behavior of different communication schemes for multi-robot systems.

Our model assumes a fair level of sophistication at the software systems level. For instance, the distributed nature of the application chains, and the need for application migration when dynamically reconfiguring. There has been recent work to use advanced software mechanisms to support autonomous robot applications. For instance, Player [8] is able to deploy software components in a distributed fashion. The CARMEN [11] software package also relies on an event-based communication mechanism (IPC) allowing flexible deployment of its software components. The MARIE project [3] is con-

cerned with building middleware to connect various robot software systems. They accomplish this by using middleware [13] capable of many advanced software systems techniques. Although, all of these packages offer some subset of the mechanisms needed to support the dynamic allocation and adaptation, no one of them can currently support it fully. Moreover, even though some of the mechanisms for distributed deployment exist, the process of making the actual deployment and configuration decisions is not automated as is left to the programmer/operator.

The idea of efficiently deploying computations in a distributed architecture has been proposed before. From a performance standpoint, there has been long standing work in mapping parallel computations onto multiple processors [12]. Taking energy into account, there has been work investigating the use of dynamic voltage and frequency scaling (DVFS) to reduce processor power consumption in distributed real time systems [10]. From an application drive perspective, a solution for efficient distributed speech recognition as been addressed [5], as well as the use of service deployment in wireless hotspots to increase the capabilities of handheld devices [15]. Support for the use of application level adaptations has also been provided by prior work which has quantified possible benefits [2], [6].

VII. Discussion

Rather than restricting the sensing or motion of autonomous robots for energy reasons, in this work we put forth an approach for addressing robot software systems' computation and communication energy behavior. Motivated by increasing autonomy and communication in multi-robot systems, and the desired independence from specific robot platforms and architectures, this paper presents a distributed systems approach to runtime power management. Specifically, it develops and presents the AutoPower model for characterizing and manipulating the software systems that execute on robotic platforms. This model allows us to make principled decisions about the configuration of application software system, including automatically deciding where to run certain software components. It also allows us to examine the trade-offs between different network media. The model is showcased by using a multi-robot search-and-rescue mission as a guiding application. The model's ability to describe system-level energy behavior results in the capture of energy trade-offs that are not obvious, with specific results demonstrating near 57% increases in lifetime by using particular system configurations.

Since AutoPower is a first step in system-wide energy management for autonomous robot systems, there are many avenues of future research. An obvious extension to this work, which we are currently developing, is an efficient algorithm for extending the lifetime of a distributed robotics application and system via intelligent allocation and choice of communication media. In the same vein, making use of dynamic frequency and voltage scaling (DVFS) may afford even greater energy savings. Second, we are expanding AutoPower to describe richer application flows, for instance, moving from application chains to describing applications with directed acyclic graphs.

Third, we are investigating ways to handle dynamic situations (e.g. failures, network connectivity, application changes) effectively, without the need for a global reconfiguration. For instance, we could have sub-groups of robots exchange software components in order to locally optimize energy use. Finally, developing methods that treat application requirements as another variable to be optimized, rather than just a binary constraint. For example, trading off localization quality, or reducing the camera's frame-rate for energy gain.

References

[1] Bruce, J., Balch, T., and Veloso, M., "Fast and Inexpensive Color Image Segmentation for Interactive Robots", *In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000.

[2] Compton, C. and Tennenhouse, D., "Collaborative Load Shedding for Media-Based Applications", *In Proceedings of the Eighth IEEE Conference on Multimedia Computing and Systems (ICMCS)*, pp. 496-501, May 1994.

[3] C C., Lourneau, D., Michaud, F., Valin, J.-M., Brosseau, Y., Ravsky, C., Lemay, M., Tran, V., "Code Reusability Tools for Programming Mobile Robots", *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2004

[4] Das, S., Hu, Y., Lee, C.S., and Hu, Y., "Performance Comparison of Communication Protocols for Mobile Robotic Sensors", *In Proceedings of IEEE International Conference on Robotics and Automation*, May 2004.

[5] Delaney, B, Simunic, T., and Jayant, N., "Energy Aware Distributed Speech Recognition for Wireless Mobile Devices", Tech. Rep. HPL-2004-106, HP Laboratories, June 2004.

[6] Flinn, J. and Satyanarayanan. M., "Energy-aware adaptation for mobile applications", *In Proceedings of the Symposium on Operating System Principles (SOSP)*, December 1999.

[7] Fryman, J., Huneycutt, C., Lee, H.-H., Mackenzie, K., and Schimmel, D., "Energy Efficient Network Memory for Ubiquitous Devices", *In IEEE MICRO Special Edition*, September/October 2003.

[8] Gerkey, B., Vaughan, R., Stoy, K., Howard, A., Sukhatme, G., and Mataric, M., "Most valuable player: A robot device server for distributed control," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2001.

[9] Mei Y., Lu Y., Lee, C.S., Hu, Y., "Energy-Efficient Motion Planning for Mobile Robots,"*In Proceedings of IEEE International Conference on Robotics and Automation*, May 2004.

[10] Mishra, R., Rastogi, N., and Zhu, D., "Energy Aware Scheduling for Distributed Real-Time Systems", *In Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*, April 2003.

[11] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit",*In Proceedings of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2003, pp. 2436–2441.

[12] Nicol, D. and O'Hallaron, D., "Improved Algorithms for Mapping Pipelined and Parallel Computations", *IEEE Transactions on Computers*, vol. 40, no. 3, pp. 295-306, March 1991.

[13] Schmidt, D.C., Huston, S.D., *C++ Network Programming: Resolving Complexity Using Ace and Patterns*, Addison-Wesley Longman, Boston, MA, 2001.

[14] Silverman, M.C, Nies D., Jung B, and Sukhatme, G.S., "Staying Alive: A Docking Station for Autonomous Robot Recharging", *In Proceedings of IEEE International Conference on Robotics and Automation*, pp. 1050-1055, Washington D.C., May 11 - 15, 2002.

[15] Su, Y.-Y. and Flinn, J., "Slingshot: Deploying Stateful Services in Wireless Hotspots", *"In Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services (MobiSys)*, June, 2005.

[16] Walter, W.G., *The Living Brain*, W.W. Norton, New York, 1963.