3-4-2008

# Behavior-Based Power Management in Autonomous Mobile Robots

Charles A. Fetzek

# Behavior-Based Power Management In Autonomous Mobile Robots

THESIS

Charles A. Fetzek, First Lieutenant, USAF

AFIT/GCE/ENG/08-05

**DEPARTMENT OF THE AIR FORCE**
**AIR UNIVERSITY**

***AIR FORCE INSTITUTE OF TECHNOLOGY***

**Wright-Patterson Air Force Base, Ohio**

AFIT/GCE/ENG/08-05

# Behavior-Based Power Management In Autonomous Mobile Robots

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

Charles A. Fetzek, B.S.C.E.

First Lieutenant, USAF

March 2008

AFIT/GCE/ENG/08-05

# Behavior-Based Power Management In Autonomous Mobile Robots

Charles A. Fetzek, B.S.C.E.

First Lieutenant, USAF

Approved:

| | |
|---|---|
| /signed/ | 4 Mar 2008 |
| Dr. Gilbert L. Peterson (Chairman) | date |
| /signed/ | 4 Mar 2008 |
| Dr. John F. Raquet (Member) | date |
| /signed/ | 4 Mar 2008 |
| Maj Michael Veth, PhD (Member) | date |

AFIT/GCE/ENG/08-05

## *Abstract*

Current attempts to prolong the life of a robot on a single battery charge focus on lowering the operating frequency of the onboard hardware, or allowing devices to go to sleep during idle states. These techniques have much overhead and do not come built in to the underlying robotic architecture. In this thesis, battery life is greatly extended through development of a behavior-based power management system, including a Markov decision process power planner, thereby allowing future robots increased time to operate and loiter in their required domain. Behavior-based power management examines sensors needed by the currently active behavior set and powers down sensors not required. Additionally, predictive power planning is made possible through modeling the domain as a Markov decision process in the Deliberator. The planner creates a power policy that accounts for current and future power requirements in stochastic domains. This provides the identification of the ability to use lower-power consuming devices at the start of a goal sequence in order to save power for the areas where higher-power consuming sensors might be needed. Power savings are observed through four simulated robots—no power management, lenient power management, strict power management, and predictive power management—in two case studies: 1) Low sensor intensity environment where robots wander randomly while avoiding obstacles and 2) High sensor intensity environment where robots are required to execute a series of tasks. Testing reveals that in a real life scenario involving multiple goals with multiple sensors, the robot's battery charge can be extended up to 96% longer when using behavior-based power management with predictive power planning over robots that only rely on traditional power management.

## *Acknowledgements*

First, a big thank you to Jeff Duffy for the use of his source code and all his code-related help and teachings.

Also, thank you to my advisor, Dr. Bert Peterson, for his support and advice—even when I decided my first thesis topic was going nowhere so I changed topics a few months before it was due.

Most importantly, thank you to my wonderful wife for putting up with days and days of me in the house but not "in the house," and to my outstanding son who, even though he's too young to understand why daddy spends all that time closeted in the other room, always makes me laugh when I come out for a break.

Charles A. Fetzek

## *Table of Contents*

Page

## *List of Figures*

# *List of Tables*

## *List of Abbreviations*

# Behavior-Based Power Management In Autonomous Mobile Robots

## I. Introduction

Autonomous mobile robots are becoming more prevalent in today's society. Since the late 1960's, much research has been devoted to the study of advanced artificial intelligence techniques. It should come as no suprise that robots have been "working" in factories building our automobiles for quite some time now. Robots are now also becoming less of a character in a science fiction story and more of a reality. Law enforcement agencies are making more frequent use of mobile robots to examine potential bomb threats in crowded, populated areas [49]. The Department of Defense is performing research into options for robots on the battlefield of the future. The use of robots allows military commanders to subject a lifeless machine to risk rather than a number of human lives [70]. Mobile robots are usually mankind's first explorer into a new world as well. NASA frequently sends robotic "explorers" onto the surfaces of our neighboring planets [28].

All the advantages of mobile robots do come at a price, however. In order for a robot to be truly effective, it needs to be free from an outside power source and rely on its onboard batteries. Of course, the robot could also utilize an expensive device for recharging its onboard batteries, such as an array of solar panels, but this solution will not be practical for most applications on a realistic budget. A robot, by its very nature, must accomplish its mission within the timeframe provided with the amount of power in its batteries. Therefore, any mechanism that provides longer-lasting batteries, or cuts down on the amount of power that the robot consumes will enable the robot to last longer in the field and accomplish a wider array of tasks.

This thesis presents a mechanism that extends onboard battery life by providing a measure of power awareness coupled with each robotic behavior.

Behavior-based power management is achieved in a reactive robotic architecture through the use of behavior representations. The representation of the current, active behavior set encapsulates the sensors that are needed for each included behavior. By traversing the tree of represented behaviors, it is possible to determine which onboard sensors are not required. Any sensor not needed for the current, active set can then be immediately be powered down. This power management scheme also provides a low power mode once the power source falls below a certain threshold. In this mode, high power consuming onboard devices are forced off in an effort to save enough power to finish the current task or travel back to a recharge station. If the robot has multiple sensors that provide similar functionality (e.g., a laser and sonar array are both capable of range detection), the higher power consuming sensor would be powered down in favor of the lower power consuming. Additionally, reasoning about the problem domain as a Markov decision process (MDP) [52] in the highest layer of the architecture creates a high level "power plan". With this plan, a robot can, for example, conserve power at the beginning of a task sequence by using low-power consuming devices in anticipation for the end of the goal set where higher-power (and hence, higher fidelity) sensors are required.

The remainder of this chapter presents a high-level outline of the information that will be covered in this thesis including an overview of the specific problem statement. Section 1.2 presents a general introduction to the main concepts that will be necessary to comprehend during this research. The goal of this investigation is then discussed. There are a few assumptions that must be made in order for this project to function and these are covered in Section 1.4.

## *1.1 Problem Statement*

There are existing techniques currently in use in the robotics and general electronics fields that provide some measure of power conservation. Certain modern electronics can lower their operating frequency when it is detected that they are not in heavy use. Lowering the operating frequency lowers the total power consumption, and in real-time systems this savings has been shown to be 20% - 40% [51]. Other devices can power down in to a "sleep mode" when it is determined that they are idle. In this state, they use minimal power–only enough to run their systems until they are needed to power back up to process information. Simunic [60] presents an algorithm that causes components in a laptop computer to power down while idle which realizes 58% less overall power consumption. In either of these power-saving techniques, there must be a trade-off between how long the device must be inactive to be considered idle, the frequency of idle states, and the amount of time needed to transition between the sleep state and the active state. The transition between states is not instantaneous. In systems were the device is used with regularity, the sleep state criteria may have to be changed so that it does not power down every time between use for a brief time when it is just going to power back up again momentarily.

Specifically, in the robotics domain, there have been efforts to save power by careful choice of the robots traversed path. The motors that propel the robot around the world are known to be heavy consumers of energy, so efficient path-planning can provide a degree of energy efficiency. This entails the robot traveling at the most efficient speed for the terrain, avoiding unnecessary direction changes and unnecessary inclines. Mei [39] has found up to 51% energy savings in an open space while utilizing efficient path planning and motor speeds. Again, this provides a measure of power conservation, but robots that primarily loiter in one place for long periods of time gain little from this technique.

This thesis seeks to fill a void in existing power savings techniques by implementing a behavior-based power management system on an autonomous mobile robot. The proposed system is coupled with the robot's reactive behavior architecture, and hence function seemlessly on any robot running it. It has the additional benefit of being completely transparent to any existing power savings techniques used by the robot, like those described previously. The following section provides a high-level overview of the key concepts needed for understanding of this project.

## *1.2 Key Concepts*

The behavior-based power management system is integrated into a reactive architecture. Specifically, it is developed in a behavior-based reactive architecture using the Unified Behavior Framework (UBF) [68]. The power management system is transparent to any form of dynamic voltage scaling or other power saving technique already in use by the system. The following subsections provide outlines of the concepts of reactive architectures and power management techniques.

*1.2.1 Reactive Architectures.* Reactive architectures are typically described as Sense, Act controllers [15]. This means the robot senses the outside world, and passes that data to a reactive controller. The controller chooses an action based on the parameters it sensed from the outside world. This system does not have a formal method for planning. The robotic architecture that includes a formal method is described in Section 2.1.1. Reactive architectures can be constructed with low-level behaviors that react quickly to stimulus and higher-level behaviors that use the lower-level's reactions. This idea of layering behaviors in such a way where the lower levels are completely independent of the higher allows the architecture to construct complex behaviors using simple behaviors as building blocks. However, situations in large, complex domains or where the robot is to perform a list of goals which would require careful planning obviously are not advisable in a strictly reactive architecture. Additionally, because reactive architectures typically contain a hardcoded library of behaviors, it does not lend itself well to code reusability for the developer when injecting the robot to a new domain. This means that reactive architecture-based robots frequently function well only in a very specific type of domain.

The Unified Behavior Framework [68] combats the problem of code reuse and robot adaptability in reactive architectures. The UBF specifies a certain way for behaviors to be defined. These behaviors can then be grouped together to form a composite behavior that selects or fuses the action or actions between the behaviors through the use of an arbiter. This way, complex robot behavior can be created using simple combinations of behaviors and arbiters. A robot built on this type of reactive behavior-based architecture is much more robust and can function in a wide variety of situations simply by modifying the composite behaviors or changing the arbitration techniques. It does not require the developer to actually create new behaviors from scratch and verify that they function correctly. The developer simply redefines new composites based on behaviors that have already been tested and verified for correct functionality.

The next step is for the behaviors in the architecture to be represented as abstractions. This allows each behavior to be associated with a specific goal it can accomplish, a precondition for execution and a postcondition that occurs after execution [21]. These behavior representations, with their respective goals and conditions, allow a measure of planning to be added to the reactive architecture. The planning and behavior representations provide a foundation for construction of a behavior-based power management system.

*1.2.2 Power Management Techniques.* As stated in Section 1.1, there are a number of techniques in use to conserve energy in robots or any other electronic device, the simplest of which is to power down idle equipment [37] [60]. Complications arise when determining when a device is considered idle or not. There is a fine line to balance the power demands of the system with the overhead of actually powering down devices when deciding they are idle. If the determination for idleness is made too often, a device could be power cycled over a very brief period of time which, due to power-up requirements being greater than operating power, cancels out the overhead of actually powering down the device. Conversely, if the determination for an idle device is not made frequently enough, very little power saving may occur since the equipment will remain powered up.

Similar to powering down an idle device is the concept of Dynamic Voltage Scaling (DVS). Under DVS, devices that are idle or under light load can have their operating frequency scaled down an appropriate amount [51]. Obviously, if the device is not operating at a high frequency, the less power it will consume. Dynamic Voltage Scaling assuages some of problems associated with putting devices in a sleep state. If the device is only operating at a lower frequency instead of completely asleep, it can still process data if unexpectedly presented with it immediately after determining it was idle. It may also cost less to bring the device back up to full operating capacity as opposed to waking up a device that has been powered down.

The problem of dynamically varying the operating voltage or powering down a device can be solved by guaranteeing when a device is going to be idle. Further power savings exist by predicting when power consumption will happen in a system, given a set of goals to accomplish or a hierarchy of behaviors to execute. Utilizing a representation of a currently active behavior that encapsulates the sensors required for execution provide a guarantee of sensor idleness. The behavior-based power manager examines all sensors that are required for an active behavior set and ensures they are currently powered up. Similarly, all onboard devices that are *not* currently required for behavior execution are immediately powered down with the guarantee they will not be needed.

## *1.3 Research Goal*

The goal of this research is to create a behavior-based power management system. It is constructed on a reactive architecture and provides a guarantee that a device on the robot is going to be idle. Through this guarantee, the robot is able to completely power down onboard devices at certain times based on the behaviors that are currently executing. This research also explores further power savings in the prediction of system power consumption, given a set of goals to accomplish or a hierarchy of behaviors to execute. This is made possible through the representations of behaviors previously described and the construction of a "power planner" using a

Markov decision process (MDP) planner in this domain with power consumption as a cost. The power planner solves the MDP problem with the Stochastic Planning using Decision Diagrams (SPUDD) open source toolkit [64], as detailed in Section 3.4.1.

## *1.4 Assumptions*

It is not the purpose of this thesis to burden the user with a specific design environment and executing on a specific hardware platform. This project should be modular enough to be developed in any object-oriented programming language in any developers environment. The architecture described herein should function correctly on most robotic platforms; however, the platform must support software control over power cycling of the onboard devices. The underlying principle that is of tantamount importance to this project is that the robotic platform being developed on supports the powering on and off of devices and sensors through software control. This is quite easy in a simulation environment where everything exists in software, but becomes a problem when executing on real, physical robots as not every device is capable of immediately turning on and off via a software command.

## *1.5 Thesis Overview*

This thesis has the following structure: Chapter I provides an introduction to the problem domain and the specifications of the research goal. It also provides a high-level summary of some of the broad topics needed for coverage of this nature. Chapter II expresses the current state of the art in research into this problem. Specifically, the chapter explains the varied robotic architectures and the use of power savings techniques in electronic and robotic equipment. It also describes power management techniques used in wireless sensor networks and reviews common methods of scheduling and planning methods. The detailed methodology for constructing a behavior-based power management system is presented in Chapter III. This chapter defines the baseline architecture, the design environment and the power management architecture as well as the test plan. Chapter IV presents and analyzes the results of testing the power management system when compared to robots without. The thesis concludes with Chapter V which summarizes the research and results and guides the reader toward future work in this area.

# II. Background

Effectively managing power consumption in any hardware or software system requires many dissimilar components working together in harmony. A mobile, autonomous platform like a robot further amplifies this challenge. A robot can easily find itself in a situation where a human operator cannot physically reach it, which would not be an ideal time for the robot to run out of power. The robotics domain thus provides ample opportunity for strong power management techniques to be exploited. Current techniques for managing power usage in robotics, whether hardware- or software-based, prove to be somewhat effective while still leaving room for improvement.

This chapter provides a background on hierarchical, reactive, and hybrid robotic architectures as well as the unified behavior framework. Following this is in exploration of the current state of the art in power management solutions in robotics and other domains with an overview of current hardware- and software-based power management techniques in robotics and wireless sensor networks. A brief discusson on scheduling and planning algorithms is provided in Section 2.3.

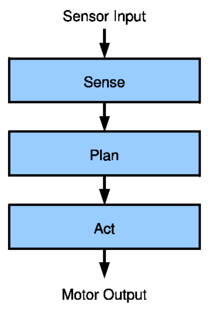
## *2.1 Behavior-Based Robotic Architectures*

This section describes the evolution of modern robots. Robots were first designed in the 1960s with a strictly hierarchical architecture. This allowed the robot to observe its domain, plan a course of action, and execute the action [48]. Later, in the 1980s, robots were developed as purely reactive entities that would execute a predetermined action given a specific stimulus [15]. This yielded a robot that could react quicker in a dynamic environment because it eliminated the time-consuming planning step. Next, researchers realized that some amount of planning is necessary for a robot to function in the real world and be able to accomplish complex tasks, which lead to the creation of the hybrid architecture [48]. This architecture allows a planning phase to be interjected into the reactive structure when processing time permits. The remainder of this section describes the specifics of these three behavior-based robotic architectures.

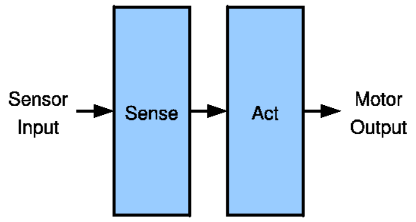
*2.1.1 Sense-Plan-Act Paradigm.* As roboticists were developing the first autonomous robots in the late 1960s, the predominant paradigm was a Sense-Plan-Act (SPA) architecture [48]. This paradigm is characterized by the clear delineation between the actions of the robot: sense, plan and act (Figure 2.1a). In the SPA paradigm, the robot first senses the environment and develops a symbolic model to define the world. After sensing is complete, the robot then plans the actions to be taken in order to accomplish the tasks assigned. It is important to note that in the hierarchical paradigm, while the robot is sensing it is not planning, and while it is planning it is not sensing. Once the plan is developed, the next action to be executed is sent to the actuators and the robot acts [48]. After the robot acts, it loops back to the sensing phase where it observes the world again with the hopeful result that the robot's action had the desired effect. The process repeats until task completion.

It is easy to see that this particular robot architecture provides a straightforward and orderly approach to developing autonomous robots. Robots were developed using this architecture in real life situations to great success [47, 50]. However, the Sense-Plan-Act paradigm's shortfall is with real time reactions. The time it takes a robot to plan out an action before acting is simply too long for many reactive behaviors. A robot using the SPA paradigm in a highly dynamic domain can easily find the state of the world quickly changed in the time it takes to decide what action to take next–rendering the decision obsolete before it was even made. This lead to the development of purely reactive architectures which remove the planning stage, thereby creating robots with quick reaction times in forever-changing environments.

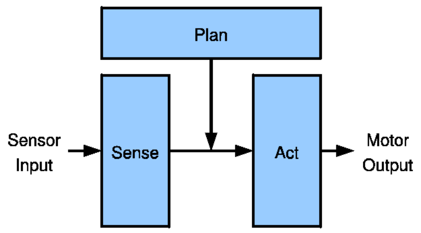
*2.1.2 Reactive Paradigm.* In the late 1980s, Brooks published his work on the reactive paradigm [15]. His architecture for robotics took planning out of the Sense-Plan-Act paradigm and replaced it with a reactive architecture of Sense Then Act (Figure 2.1b). Brooks modeled his architecture on biological entities rather



(a) Hierarchical Architecture



(b) Reactive Architecture



(c) Hybrid Architecture

Figure 2.1: Robotic Architectures

than high-level symbolic reasoning. He showed that living creatures can have simple reflexive reactions and still accomplish complex tasks.

In the reactive paradigm, low-level behaviors are developed to provide quick reactions to certain sensor inputs. For example, an initial behavior for a wheeled robot might be to avoid colliding with obstacles. Another behavior can then be added on top of the first, providing the robot with additional functionality. For example, a behavior for the robot to drive around in a random pattern can be added. These two behaviors would allow the robot to explore its surroundings while avoiding collisions. As behaviors are created higher up in the behavior architecture, they can inject information into the lower layers for them to act on without the lower layers even aware that it is coming from a layer higher in the architecture. Brooks calls this the Subsumption architecture [15].

The reactive paradigm proved to be quite robust and also quite prevalent throughout the 1990s [48]. Careful selection of the behaviors underlying the layers caused a robot to achieve a variety of complex tasks without an actual planning phase as in the Sense-Plan-Act paradigm. However, robots developed with a reactive architecture are hardwired for a specific domain and specific set of executable tasks. A reactive robot cannot be easily removed from one domain by the developer and placed into a new domain with new goals without greatly modifying the underlying behaviors. Consequently, there are several behavior-based reactive control architectures, each of

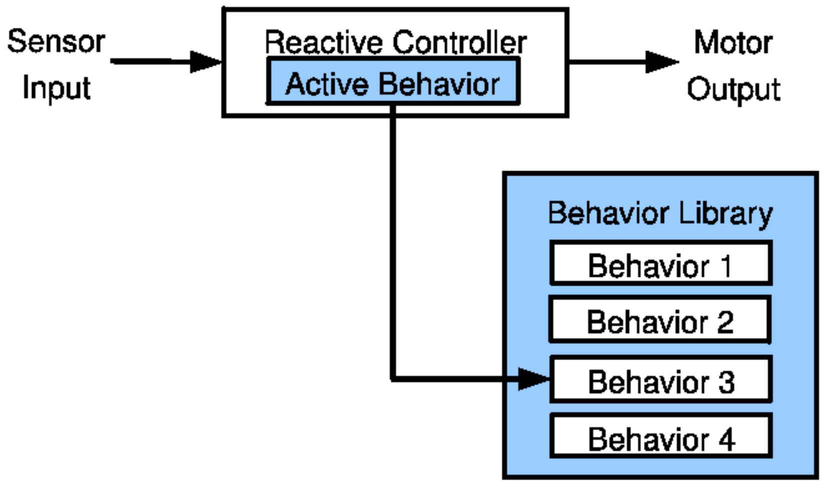


Figure 2.2: Unified Behavior Framework Architecture

which excel in a particular domain [8] [15] [18] [31] [34] [54]. This problem is assuaged by the Unified Behavior Framework.

*2.1.3 Unified Behavior Framework.* The behavior-based robotic architecture paradigms described thus far rely on a single behavior-based system that is hardcoded during robot development. This results in a robot that is not as adept at cross-domain execution, as its repertoire of actions to choose from is specifically tailored to the initial domain and goal set. The robot is forced to conform to a single reaction-based architecture which—while suitable in the initial domain—might not provide the action needed in a new situation. Thus, several behavior-based reactive architectures have been developed and proven to function successfully in their specific domains. The Unified Behavior Framework (UBF) provides a means of selection between techniques from varied behavior-based reactive architectures dynamically at execution time [68].

The UBF defines a standardized way of describing simple behaviors following proper software engineering techniques [68]. Figure 2.2 shows how the controller chooses between multiple behaviors inside a library in real-time since each behavior is described in a similar fashion. As in the Reaction Action Packages described by Firby [23], the UBF allows the controller more freedom in its action selection under any given situation presented through its sensors. Having multiple behaviors to select an action from allows a robot utilizing the UBF to much more readily function in a highly dynamic environment.

Additionally, the UBF presents a standardized way of describing arbiters for any group of behaviors. This allows groups of complex composite behaviors to be constructed out of relatively simple single behaviors. Arbiter design is completely up to the designer, but they can use such functions as highest activation, where the behavior with the highest utility value is executed, or utility fusion where each utility value is weighted and a combination of behaviors or partial behaviors are executed [68]. The UBF, however, does not provide a formal method for an incorporated planner. Planning is vital in domains that require multiple, complex, or temporally constrained goals. However, the incorporation of a planner comes with the possibility of increased time consumption.

The Unified Behavior Framework was further improved upon by abstracting each behavior in a list of behaviors (or behavior library) into their components [21]. Each behavior is described by a set of goals it will accomplish, a set of preconditions necessary before it can execute, a set of post conditions that behavior execution causes, and a set of data required for the behavior to access. Since the behaviors are now represented as abstractions, composite behaviors can be constructed dynamically during execution. This provides a robot with many more behavior choices to accomplish its goals. As will be shown in Chapter III, this dynamic architecture provides the groundwork for behavior-based resource management.

#### *2.1.4 Hybrid Architectures.*

By the late 1990s, researchers realized that reactive agents worked well in dynamic environments without long-term, complex task planning and hierarchical agents worked well in static environments to capitalize on their ability to plan. Neither paradigm was suited for an autonomous agent in the real world where change can occur rapidly and tasks are complex.

The solution was the hybrid architecture. The hybrid paradigm merges the best of the hierarchical and reactive paradigms. Agents using the hybrid paradigm function thus: Plan, then Sense and Act (Figure 2.1c). Essentially, the agent first plans out the behaviors needed to accomplish a task, and then executes them in a reactive way. It has the ability to process planning in parallel with reactive execution of current tasks. The planner module can also interrupt the current execute to interject a new behavior, if needed [48].

Virtually all autonomous vehicles have some form of one of the three architectures described here as their underlying framework. These architectures allow robots to flourish in dynamic or static environments and with complex or simple goals. These architectures do not, however, provide any means of managing the robot's power consumption. This thesis presents an architecture that incorporates reliable power management for autonomous vehicles without sacrificing any of the advantages inherent in the original architecture.

## *2.2 Review of Power Management Techniques*

Autonomous mobile robots appear in a myriad of applications. A consumer can purchase a reasonably "smart" autonomous vacuum cleaner for their home (Figure 2.3a) [29] or play with a robotic toy dog (Figure 2.3b) that reacts to the consumer's commands [63]. However, autonomous robotic vehicles have a critical dependency: by their mobile nature, they cannot be tethered to a power source. Each robot must rely on its own internal batteries and hence efficient power management becomes an important design issue.

There are only a few approaches to power management currently employed in mobile electronic equipment. This section presents a brief description and overview of these techniques. Efficient use of limited battery energy can be hard-wired into the design of the hardware as well as the design of the software [14]. Many of today's modern operating systems include provisions built-in for power management that the robot can utilize [10]. There are also several process scheduling and voltage scaling

(a) The Roomba Autonomous Vacuum Cleaner

(b) Sony's Aibo Robotic Dog

Figure 2.3: Two examples of robots available for consumers

algorithms in use to determine the most efficient way to execute programs [5] [6] [51]. The remainder of this section provides further details on the techniques mentioned.

*2.2.1 Hardware Design.* Processor speed and voltage demands are ever-increasing while the physical size of the processor is ever-decreasing. As computer processors get faster, their power requirements become the strongest limiting factor in their future performance on a mobile platform such as a robot [66]. One possibility to combatting this problem is to handle the power management at the level of the chip design itself. As the processor is designed, efficient power management is incorporated. Brooks [14] proposes the *Wattch* framework for exactly this purpose. Wattch provides architecture designers with toolsets for use while designing the hardware that allows for the testing and analyzing of power demands. The open source toolset, SimpleScalar [59], simulates modern processors and is often used as a design and debugging environment in research and industry [4] [16]. As such, Brooks modified SimpleScalar's structure to include the Wattch framework. Brooks shows that since the power usage is optimized at this low-level, early in the design of the system, power efficiency is increased and initial architecture development time is decreased

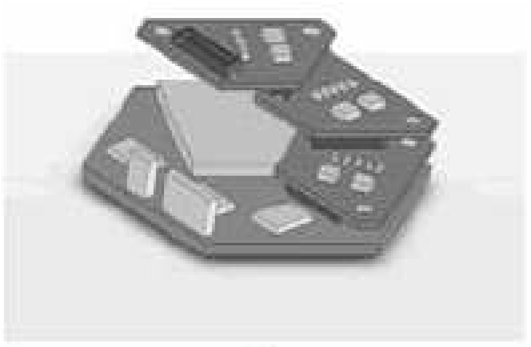


Figure 2.4: The circuitry behind D'Souza's TerminatorBot, showing the spiral design of the removable FPGA modules

since future redevelopments due to power issues are avoided. This allows the power management to be handled at a level much lower than the Operating System kernel, and reduces the OS requirements, allowing it to operate more efficiently.

*2.2.2 Hardware Removal.* While it is possible to save power through efficient design of the chipset, sometimes it is more cost-effective to simply remove hardware components that use too much power. A novel approach of a field programmable gate array (FPGA)-based, resource constrained robot is outlined in D'Souza's work [20]. D'Souza describes an FPGA-based *Morphing Bus* specifically used in small, about 3 - 5 inch diameter, robots which could also be scaled to use in larger applications. Figure 2.4 depicts the concept of modular FPGA-based sensor packages that are connected in a spiral, in series. His use of FPGAs in each module of the robot allows the user to decide which sensor packages to connect to the system at execution time. The user in the field could decide, for example, to connect an IR camera and a sonar locator but leave disconnected the GPS receiver and laser range finder. This allows the robot to save power by only using the components that are needed for the current task with the obvious drawback of the user needing to physically connect or disconnect the appropriate components.

*2.2.3 Dynamic Power Management.* Perhaps the most prevalent energy conservation technique is "turning off" power scavenging systems that are not needed.

This technique of providing enough embedded systems to complete the task with the minimum amount of power through eliminating temporarily unnecessary components is calling dynamic power management (DPM) [10]. Consider a swarm of small, unmanned aerial vehicles used for loitering and surveillance over a large area inhospitable to humans (such as over a forest fire) [37]. Marinoni's design for an embedded controller for autonomous flight control incorporates the power requirements of each I/O peripheral in the DPM algorithm. Most components on the vehicle can be turned on or off through through a direct digital line which further decreases the energy needed to power up or power down. A significant amount of the energy in Marinoni's vehicles is conserved through efficient management of the communication channel. Because the vehicles travel in groups, there will always be some transmission and receipt of messages between robots. However, as the vehicle loiters overhead it can power down the radio module so as not to waste energy on idle listening. However, in Marinoni's design, the onboard devices' power states as well as the internal CPU are controlled through a separate microcontroller that takes into consideration the anticipated operating frequency to execute the current task. This creates more overhead than a strictly behavior-based system and does not provide a guarantee of device availability at the correct time, nor device power down when truly idle. Efficient communication protocol design also designates the power transmission value, and the retransmit time to ensure packet collision avoidance and error-free transmissions with great efficiency.

Motorized hardware can use up to half of the energy in a robotic vehicle [40]. For example, the Pioneer 3DX robot by ActivMedia can use between 2.8 -10.6 Watts just for turning the wheels to produce locomotion. Thus there is a large potential for energy savings for efficient path planning and velocity control. Avoidance of frequent speed changes and excessive milage beyond the direct route are two methods of DPM to create efficiency in robotic kinematics [38].

Robots acting in a swarm or group present their own unique set of power management challenges. In a group environment, the energy efficiency of the group as a whole, as well as each individual robot, must be taken into consideration. The initial deployment of the robotic swarm can play a large role in efficient power management [41]. Robots must deployed with enough density to cover the area needed, while at the same time meeting timing requirements. Mei [41] shows that a proper deployment technique of a swarm can cut power requirements for the group as a whole by up to 32 percent.

*2.2.3.1 Dynamic Voltage Scaling.* Dynamic Voltage Scaling (DVS) is another type of DPM found in a large number of today's efficient power designs. DVS is a way to dynamically raise or lower the voltage and/or operating frequency a processor (i.e., alter its speed) in order to save power. This can make the processor enter an ultra low power, or "sleep" state. Aydin [5] shows that real-time systems are one area that DVS works particularly well. Typical variable voltage algorithms use the worst-case execution time when determining the scheduling of processes in an effort to use the minimum amount of voltage possible and still complete the processes within the temporal constraints. Aydin took this one step further by introducing a real time heuristic that constantly monitors actual CPU computation time and adjusts voltage requirements. Further, a speculative component of the heuristic is provided to predict when idle periods in the processor will occur. Using each of these techniques as opposed to the static worst-case execution time algorithm can save an average of 50 percent of the energy consumed [6].

The next step for Dynamic Voltage Scaling is to incorporate it directly with the scheduler in a real-time operating system. This integration is critical since real-time tasks are dependent upon temporal execution constraints as well as power requirements. Allowing the DVS scheduling algorithm to be integrated into the operating system's real-time scheduler has shown to improve energy efficiency almost to the lowest theoretical energy consumption [51]. Pillai shows that the *real-time* DVS (RT-DVS) algorithm has proven to save 20 - 40 percent of energy consumption in embedded real-time systems. Testing of various versions of RT-DVS was performed by Kim in a thorough, simulated environment, SimDVS [32]. Each RT-DVS algorithm was subjected to rigorous testing in the same domain to get an accurate measure of energy efficiency comparisons. He shows that utilizing the best RT-DVS schemes result in power consumption only an average of 9 - 12 percent above the theoretical minimum power usage.

*2.2.4 Wireless Sensor Networks.* Wireless sensor networks are also a power-constrained embedded system. They offer robust and fault-tolerant solutions to networking problems as varied as wildlife habitat monitoring to node localization [2] [19] [22] [35]. Wireless sensor networks are usually implemented with swarms of small microcontrollers coupled with a sensor package and/or a wireless communication device. The appeal for wireless sensor networks comes from the fact that they have the potential to be spread out across a wide area and loiter for long periods of time. However, this ability comes with a cost: the individual network nodes are useless once their onboard battery is depleted [46]. Consequently, there have been many endeavors into power management for wireless sensor networks.

Power savings can start at the lowest level of design—the physical layer [56]. This incorporates both the actual radio wave transmissions and the hardware needed. Efficient design of a physical layer protocol for communication between wireless sensor network nodes can provide power savings, for example through exploiting the clustering of individual nodes. Shih [56] shows that multiple nodes in a group can function as a virtual single node. Information passed between each node in a cluster is redundant and thus, data only needs to flow from one cluster to another. Hui [27] presents a unique approach in which power management is achieved through a sentry-based approach. Each node is grouped into a sentry/non-sentry status. The nodes that are non-sentries are able to shut down until called back to service by the sentries. This allows power savings for the group as a whole.

Additionally, not all media access control (MAC) protocols are as efficient as others. Complex, handshake-based MAC protocols such as carrier sense multiple access with collision avoidance (CSMA-CA) increases latency between transmissions with their additional computations over time or frequency division multiple access (TDMA or FDMA) and hence use more power as well. In his article [71], Ye designs a custom sensor-MAC (S-MAC) in order to save power and still retain some of the benefits of a complex MAC such as CSMA-CA. S-MAC includes provisions for automatic duty cycle with periodic sleeping and adaptive listening based on current traffic flow. Overhearing avoidance and inter-node message passing are also implemented in S-MAC.

The benefits of dynamic power management described in Section 2.2.3 can also apply to wireless sensor networks. Specifically, Sinha [61] compares various DPM algorithms, including utilizing dynamic voltage scaling, in a wireless sensor network environment. The results show that using an execution rate for the microprocessor timed the same as the overall average workload achieves the maximum power savings.

Parts of all the techniques previously described are captured by Zheng's power management scheme for wireless ad hoc networks [72]. This scheme achieves on-demand power management by monitoring traffic flow and density. Each node can observe traffic conditions and if no or limited traffic is flowing through it, the node will power down and rely on the inherent redundancy of the wireless network to maintain connectivity for the other nodes. As network management packets are sent through the network, it can trigger the sleeping nodes back into service, since these packets could signal a change in the network topology. Utilization of this framework was shown to increase energy efficiency over standard ad hoc networks by a factor of about 1:5.

Modern robots are designed with little to no inherent power management such as the techniques described here. It is common for robot designers to rely on whatever power management that might be built into each hardware component of the robot. For example, the operating system of the robot might include provisions for Dynamic Voltage Scaling to allow idle hardware to power down to sleep mode. However, the robot architecture itself does not include energy savings techniques such as dynamic voltage scaling or dynamic power management. This thesis incorporates power management directly into the robotic architecture by using a form of behavior-based dynamic power management.

## *2.3 Scheduling and Planning Algorithms*

A related topic to Dynamic Power Management is the balance between power cost and sensor accuracy. Ideally, a sensor would have a very high fidelity with low power cost, but in real life this is rarely the case. Trade-offs occur when developers require a high resolution sensor for a long period of time—potentially draining the power source before task completion. Similarly, dynamic voltage scaling algorithms require specific details on when to increase or decrease the voltage to hardware.

In order for the dynamic power management and voltage scaling techniques described in Section 2.2.3 to work efficiently, a reliable scheduling algorithm is needed. If the processor enters its sleep mode too frequently and for short periods of time, there will be higher energy costs to pay starting it back up [11]. Additionally, the time wasted powering the processor down and back up unnecessarily could potentially cause the processes to miss real-time timing constraints. Shin [57] introduces low power fixed priority scheduling (LPFPS). LPFPS works by dividing the power consumption problems into two cases: one case where all tasks have finished executing on the processor and one case where all but one process has finished executing on the processor. In the first scenario, LPFPS powers down the processor into sleep mode for a period of time slightly less than the amount of run time needed for the top process on the list. It is slightly less time because there is a small amount of overhead needed to bring the processor out of sleep mode. In the second case, the processor's speed can be controlled by DVS to allow the most energy efficient output to execute the single task, and then power down into sleep mode in preparation for the next incoming process.

Benini [11] proposes an alternative scheduling algorithm that abstracts the process queue into a Markov, rather than the common heuristic, decision process. Using a Markov decision process (MDP) allows the scheduling problem to be treated as a common stochastic optimization process. He goes on to show that this type of policy optimization problem can be solved exactly within the framework he described in polynomial time. This does not mean that heuristic-based scheduling algorithms do not show any promise. In fact, a heuristic-based real-time scheduler was shown to work to great effect by Mejia-Alvarez [45]. He uses a rapidly executing scheduler based on approximate solutions to the knapsack problem. This heuristic proved to yield overall energy savings of approximately 25 - 30 percent.

Xian [69] further improved upon the real time Dynamic Voltage Scaling algorithm. He shows that scheduling across multiple processors is made more efficient by constructing probabilistic distributions of the worst case execution times of the tasks. When tested on multimedia applications, stereovision calculations and synthetic algorithms, his scheduling algorithm for multiple processors can show energy savings from 19% - 30%.

It is important to note that these solutions focus on voltage scaling power management for the computer and while many robotic designs include a planner, the robot's planner does not incorporate power considerations like those described here. Robotic planners are strictly goal and task-based. The stochasticity of a real-world environment makes planning based on power requirements a nontrivial problem. However, utilization of an MDP planner to reason about the robot's domain with sensor power consumption is a potential solution.

*2.3.1 Markov Decision Process.* An MDP is expressed as a set of states, actions, transitions between the two, and associated costs. Specifically, an MDP is defined as the four-tuple, (S, A, P, R), where S is the set of states, A is the set of actions, P is the probability that action $a$ in state $s$ at a certain time, $t$, will lead to state $s'$ at $t$+1, and R is the reward or cost associated with that transition [52]. The MDP is solved by maximizing the overall reward or minimizing the overall cost. MDPs assume that all information about past states are captured in the current state, which means that knowledge of the history of states is not needed to predict the future from a single state. This is defined as the Markov assumption. Because MDPs make use of the Markov assumption, they are used widely in the Artificial Intelligence community for solving stochastic problems where the sequence of previous states will not effect the current decision [12]. A Markov decision process will produce a policy that depicts the optimum action to take at any given state. Planning with an MDP occurs when the state-space and action-space is examined for the current state, and the action producing the highest reward (or lowest cost) is chosen. Because the resultant policy solves the MDP problem for the entire state- and action-space, an optimal action selection is guaranteed. Solving an MDP and producing a policy graph can be done with several, well-known algorithms such as linear programming, policy iteration, and value iteration [33]. A python script to solve MDPs is included in Russell and Norvig's definitive Artificial Intelligence textbook [55], and several open source tools have been developed to solve MDPs including SPUDD [64], and pomdp-solve [17]. The stochasticity of power planning leads to a MDP-type problem, as is shown in the following chapters.

## *2.4 Summary*

This chapter presented an introduction to the hierarchical, reactive and hybrid robotic architectures as well as the Unified Behavior Framework and explained that existing architectures do not have inherent power conservation techniques. State of the art power management techniques for robots and wireless sensor networks were also explored for their possible incorporation into a robotic architecture. This chapter showed how careful hardware design and configuration can produce a more energy efficient robot and how use of dynamic power management and voltage scaling can help keep isolated robots and wireless sensor networks running longer on their limited battery life. Planning and scheduling algorithms, specifically relating to DVS were also discussed with their importance relating to real-time systems and the potential for inclusion into a robotic architecture. The next chapter outlines the proposed robotic architecture that includes behavior-based power management.

# III. Methodology

The basic robotic architectures—hierarchical, reactive, and hybrid—are currently prevalent in the research community and industry, and they do not have inherent power management. To prolong robot functionality, proper power management techniques should be included in a robotic architecture. The inherent nature of a robot relying solely on its onboard battery demands that maximum energy efficiency be obtained. It is easy for a robot to find itself in a physical location that a human cannot reach when the battery dies. For example, all of NASA's Mars rovers must have a built in system for power management techniques although the rovers enjoy the advantage of a rechargeable energy source [53]. However, various power management techniques exist that can be employed in the baseline robotic architecture of autonomous vehicles.

This chapter describes, in detail, the methodology for adding robust, tailorable power management functionality to a behavior-based robotic architecture. Specifically, Section 3.1 presents the motivation and purpose of the project, followed by the project design starting with the initial architecture. The specifications of the power management architecture are presented in Section 3.4. The simulation environment is then discussed and the chapter concludes with the testing strategy.

## *3.1 Purpose*

The goal of this research is to provide a reliable, robust, and tailorable system for managing the power and resource consumption of an autonomous mobile robot. As shown in Chapter II, there are many available opportunities for power savings in robotic systems. Many of the most modern robots rely on energy conservation from techniques such as efficient path planning or well-scheduled motor power and processor operating frequency and voltage [13] [42] [43] [44]. While energy conservation algorithms such as these can produce significant power savings, there are still disadvantages. For example, path planning-based power savings do not have much effect on a robot that remains primarily motionless. A robot designed for high loiter times, such as those in a mobile wireless sensor network, still need other techniques designed into their architecture to ensure their other, non motion-based abilities are energy efficient. The same argument is true for robots relying on efficient scheduling of the motor power commands and processor speed.

It is clear that another approach is needed to produce energy efficient robotic systems. The ideal power management system would be built in to the robotic architecture itself, be it reactive, deliberative or a hybrid. Further, it should function regardless of the platform it is operating on, and be customizable enough so that a developer can use it on any of their systems. Finally, it should ideally function "right out of the box" meaning that the developer doesn't have to spend man-hours tweaking many parameters to ensure the maximum energy savings and should instead have one or two options to choose from in order to function. The remainder of this chapter goes into detail on just such a power management system.

## *3.2 Design Overview*

This section presents the steps taken to design an efficient power management system for autonomous mobile robots. A base robotic architecture is chosen to lay the foundation for an energy efficient scheme. A suitable platform for modeling and simulating is also chosen, as well as the testbed robotic platform. At each step of the design process, the goals of reliable, robust and tailorable are maintained.

The power management solution itself consists of a novel behavior-based approach. This approach utilizes behavior representations that encapsulate (among other critical information) the sensors required for behavior execution. Each time a new behavior representation hierarchy is activated, a list of required devices is produced. The sensors that are not required are immediately powered down with a guarantee they will not be needed. There is no calculation required for sensor idle time or dynamically scaling the voltage to a specific operating frequency. In fact, the voltage is scaled to zero for sensors that are not needed, creating maximum possible energy savings. Additionally, the architecture enables a "critical power mode" when the onboard power source falls below a certain threshold. This mode causes the higher power consuming devices to be powered down in favor of lower power consuming (at the potential cost of decreased sensor resolution). This allows the robot to extend its battery life in an attempt to execute a final task before battery depletion or to reach a recharge station.

For the proposed power management system, it was decided to start with the groundwork of a reactive robotic architecture. As shown in Section 2.1.2, the reactive architecture provides a robot with reliable, stable repertoire of behaviors to select from at execution time based on environmental inputs. Assuming the library of behaviors is well-constructed, a reaction by the robot to almost any given outside stimulus is guaranteed. Adding the Unified Behavior Framework [68] to the classic reactive paradigm allows the robot to arbitrate between behaviors in its library and create composites of combined behaviors which provide it with even further capabilities at execution time. Finally, because the power management system functions according to the data demands of any active behavior, the behavior abstractions provided by Duffy [21] are used. This initial design infrastructure is further detailed in the next section.

## *3.3 Initial Architecture*

The power management system is built upon a starting substrate of the Unified Behavior Framework coupled with behavior representations with dynamic goal processing [21] [68]. One of the main advantages for using the UBF is the quick reaction time of a Sense-Act architecture with the flexibility of composite behaviors formed dynamically at run time. The UBF can also maintain a clear delineation between the three layers in the architecture (Deliberator, Sequencer and Reactive Controller, as defined in [24]) because arbitration of the composite behaviors occurs inside the controller while the other two layers function independently. The strong encapsulation of behaviors in the UBF is taken a step further with the addition of behavior representations (or abstractions). The behavior representations enable a planner to search

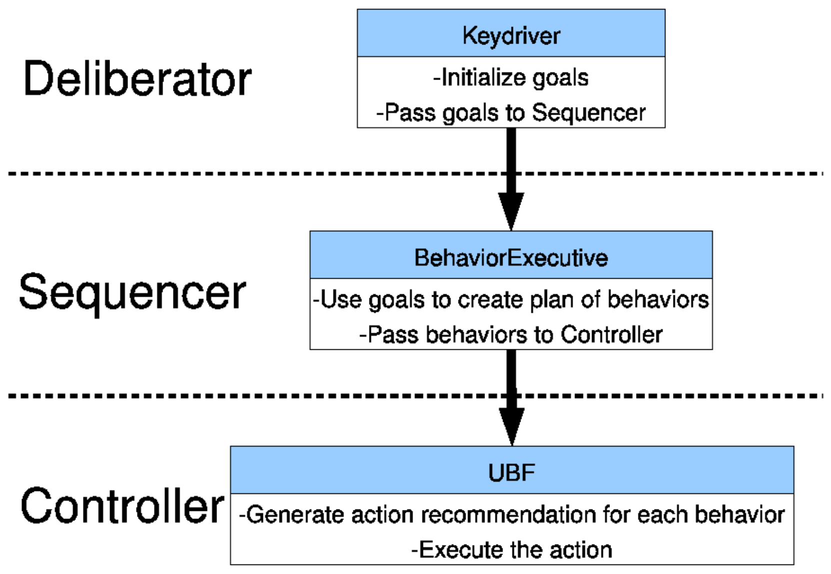


Figure 3.1: The initial architecture before the power management module is added. Note the clear delineation between the three layers

a set of behavior post-conditions and construct a behavior hierarchy that completes a given set of goals. Figure 3.1 illustrates the separation of the layers and the Unified Behavior Framework acting as the reactive controller. The following subsections describe each layer in more detail.

*3.3.1 Deliberator.* The deliberator, or planner, is the top-most layer in the architecture. The deliberator constructs a hierarchy of goals and passes it down to the next layer. Figure 3.1 also depicts the deliberator as the Keydriver. This is because the system allows for keyboard control of the robot, as well as goal planning. This provides the ability for the user to enter specific keyboard commands during execution as well as the execution of abstract goals. The deliberator compiles the entered goals into a goal set for later processing in the sequencer layer.

Upon startup of the system, the deliberator creates an initial goal that becomes the default goal for the robot to execute when no other goal set is active. A common default goal is zero forward and angular velocities. This is quite useful so that the robot does not immediately power up and drive off in persuit of another goal. The initial goal is passed down the architecture to the sequencer. Subsection 3.3.2 explains events at the lower layer. After passing the initial goal to the sequencer, the deliberator waits for keyboard commands from the user. The user at this time has a number of options. They can drive the robot manually or place the robot into one of many operational modes (for example, wander the room or drive to a specific coordinate). The user may also enter one or more pre-programmed goals. This is perhaps the most interesting aspect of the behavior representation architecture put forth in [21]. Each goal the user enters is combined into a set of goals for sequential execution. In this manner, the robot can be instructed to travel to a specific grid coordinate, then search for a specific colored object, pick it up, travel to a different grid coordinate, set the object down, then wander the room while waiting for additional commands. Because it is constructed modularly, the user can enter almost limitless combinations of goals into a goalset, each of which can be further customized during the development phase.

*3.3.2 Sequencer.* The sequencer resides in the middle of the three layers. It receives goals or goalsets from the deliberator and constructs a hierarchy of behaviors for the controller to execute. In Figure 3.1, the sequencer is referred to as the BehaviorExecutive. This is because the sequencer acts as the interface between the behavior library and the UBF in the controller layer.

The controller layer expects to receive a single behavior to execute. Because of the encapsulation of behaviors allowed by using the UBF, this single behavior can actually be a series of behaviors that are combined into a composite behavior through the use of an arbiter. From the controller's point of view, however, a composite behavior is still just a single behavior with one Action command to execute. The top priority of the BehaviorExecutive (or sequencer) is to send the active behavior set to the controller to meet the current goal. There is idle processing time after the behavior is passed down, so the sequencer can process the current goalset to develop plans to meet them. The BehaviorExecutive uses partial order planning techniques [9] to examine the list of current goals and construct a hierarchy of plans which in turn becomes a hierarchy of behaviors. In this way, the BehaviorExecutive can also determine if the list of goals has no possible plan to solve them if, for example, preconditions cannot be met. If the BehaviorExecutive did find a plan for the current goal, the behavior hierarchy is passed down to the Unified Behavior Framework inside the controller.

*3.3.3 Controller.* The lowest layer in the architecture is that of the controller. This is where an action is generated from the current active behavior and translated into actual, physical motor commands. This makes the reactive controller perhaps the most straightforward of the three layers. If the controller is passed a composite of multiple arbitrated behaviors, it only generates a single action to execute out of the group. This action could be chosen through arbitration as simple as highest activation, where the behavior in the composite that votes the highest gets to generate the action. The controller may also generate an action through a utility fusion arbiter [54], whereby each behavior in the composite calculates a vote based on an expected outcome of its action and the action with the highest expected utility value is selected. As the action is being executed, the sequencer can preempt with a new set of behaviors, and the planner can preempt the sequencer with a new set of goals. The process then repeats to completion.

The foundation that the power management module will be built in has been described. As shown, it is a well-constructed, modular design that supports interoperability between its components. The power management pieces will fit inside as another module. The coupling of the power management system to the initial architecture is detailed in the following section.

## *3.4 Power Management Architecture*

The power management architecture described herein functions using a few basic principles. First, as demonstrated in Section 2.2, there are already energy efficient techniques currently used in the industry that focus on the design or operating speed of the hardware, and the ability of voltage scaling to put idle equipment to low-power "sleep" mode. Therefore, these techniques can still be used while the behavior-based power management system is functioning. That is, this thesis proposes a system of power management that runs concurrently with existing energy efficiency algorithms. Second, the act of coupling the power management to the encapsulated behaviors has the potential to grant a degree of power predictiveness to the system. For example, the list of sequential goals the planner passes down to the sequencer can be associated not only to behaviors that enable those goals, but also to identify power requirements that can meet those goals. This way, the planner can "predict" that low power consuming sensors be used at the start of plan, in preparation for high-power demands that will occur at the end of the plan. This is accomplished by modeling the domain as a Markov decision process in the deliberator and using the resulting policy to determine the optimal power status of the sensors at any given time. Finally, the behavior-based power management system should be device independent and simple to configure. As long as the hardware functions with the initial reactive architecture, the power management system should be seemless and transparent to the robot. With these principles, a system can be created to maximize energy efficiency on a multitude of platforms with a myriad of hardware configurations.

Behavior-based power management differs from the traditional power management techniques described in Section 2.2 in that if the current active behavior does not use a particular sensor, it is immediately shut down resulting in zero power usage. This technique does not rely on a time period for the sensor to be idle before powering down, nor does it have a "low power" consuming state. The sensor is simply rendered inoperable as if it were disconnected from the power source. Upon activation of a new behavior that requires that particular sensor, the sensor is powered up and functions as normal until no longer required.

The resource management module keeps track of the current status of each hardware device. This status is an internal representation, only accessible to the resource manager. A sensor device is considered AVAILABLE if the sensor is currently subscribed, (i.e., powered on) and data is being collected from it. A sensor device is changed to status ACCESSIBLE if the device is not currently subscribed (i.e., powered down), but it *can* be subscribed to if a new behavior requires that sensor's data. Finally, a device is considered UNAVAILABLE when it is not currently subscribed, nor will it be allowed to ever be subscribed barring a change in the current power status. The reason for the distinction between ACCESSIBLE and UNAVAILABLE involving sensors that are powered down is that if the power source falls below a certain threshold, the resource manager will prevent access to devices that consume large amounts of power in an effort to extend battery life far enough for the robot to, for example, travel back to a recharging station.

The resource management system described in this section was designed and tested using a network proxy interface to connect to a simulated robot provided by the Player/Stage open source develop tool set. (The specifics of the interface communication and robot simulation of Player/Stage are presented in Section 3.5). As such, the simulated robot does not actually power up and down its sensors. Instead, this thesis makes use of Player-defined status called "subscribed". When a Player connection is initialized to a device client, the sensor must be subscribed so that Player can utilize the sensor's data. In the Stage environment, the act of subscription essentially causes the sensor to "turn on" and unsubscription to "turn off". It is important to note that on a real, physical robot, subscribing and unsubscribing to the sensors will, depending on the specific hardware, not actually power up or down the device. It will only cause the communication link to be up or down. As stated in the assumptions, this system relies on hardware that can be powered up or down through software. The Player interface already has a built in hook for device power, so it should be a trivial matter to switch from subscriptions to actual power commands to turn sensors on and off.

Reducing power consumption based on active behaviors is made possible largely in part through the architecture developed by Duffy [21] and explained in Section 3.3. The creation of representations for each behavior allows the encapsulation of not only pre- and post-conditions but also sensors required for activation. This makes it possible for the incorporation of a power management module into the architecture that checks required sensors for current (or active) and future behaviors and adjusts running sensors as necessary. In addition, because the architecture essentially transforms a list of goals into a hierarchy of behaviors, some measure of predictive power management is possible, while being constructed on top of the reactive three layer architecture previously described. Figure 3.2 depicts an overview of the new architecture, with the Resource Management module added in. The specifications of this behavior-based power management system are provided in the following subsections.

*3.4.1 Deliberator.* The addition of resource management to the initial architecture described previously in Section 3.3 provides added functionality to the Keydriver class. The keydriver inside the deliberator layer still generates a list of goals to be achieved. This can again be through user input at runtime, or goals that are preprogrammed into the system to run sequentially at startup. However, since the system now has a degree of control over the power consumption, the deliberator has the option of sending various power requirements down to the sequencer along with the set of goals. Figure 3.3 depicts the program flow in the deliberator layer. The deliberator has optional commands to send to the sequencer that specify sensors to turn on and off, besides those that the resource manager selects on its own.

Passing power commands from the higher, planning layer provides the system a degree of predictive power management, as the deliberator has a broad, overarching plan of all the goals that need to be accomplished, while the sequencer (detailed in the next subsection) focuses only on the immediate task. For example, the user entering

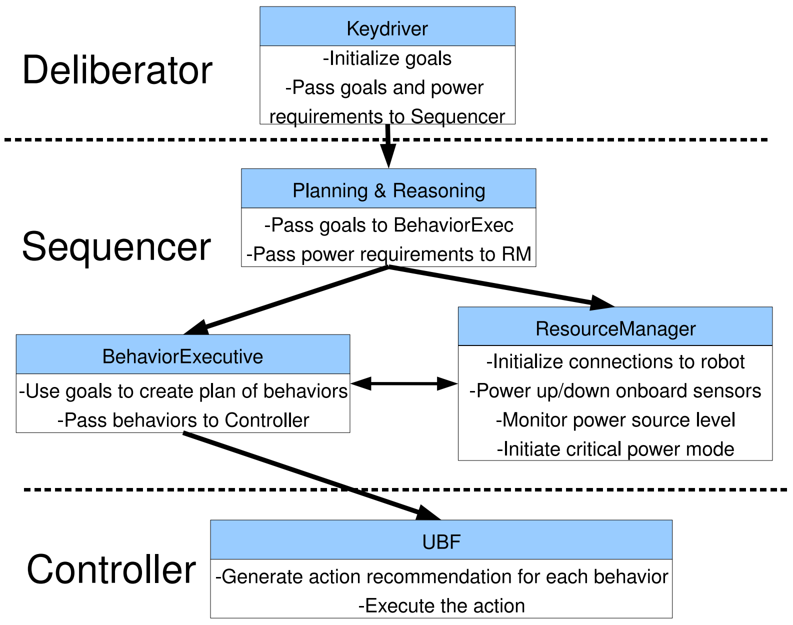

Deliberator
Keydriver
-Initialize goals
-Pass goals and power requirements to Sequencer
Sequencer
Planning & Reasoning
-Pass goals to BehaviorExec
-Pass power requirements to RM
BehaviorExecutive
-Use goals to create plan of behaviors
-Pass behaviors to Controller
ResourceManager
-Initialize connections to robot
-Power up/down onboard sensors
-Monitor power source level
-Initiate critical power mode
Controller
UBF
-Generate action recommendation for each behavior
-Execute the action

Figure 3.2: The three layer architecture with power management included. Power management occurs in the sequencer, parallel with the BehaviorExecutive. A top level Planning and Reasoning class is added to handle communication from the Deliberator to the BehaviorExecutive and Resource Manager.

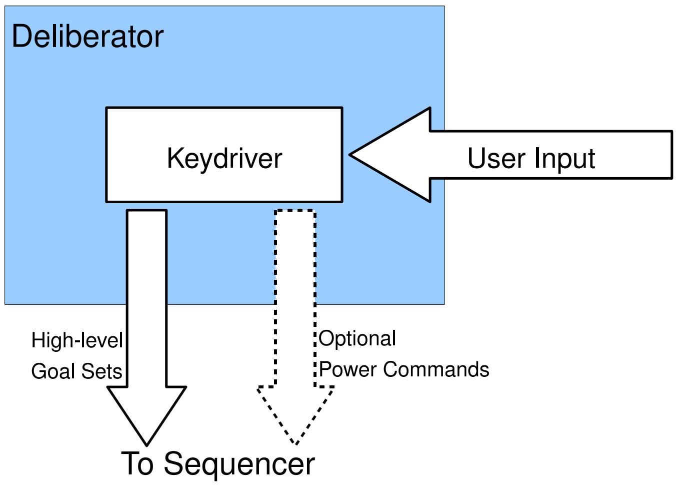


Figure 3.3: Detail of the Deliberator layer. User commands are received from outside, goal sets and power commands are sent down to the Sequencer.

a sequence of goals may have first-hand knowledge about the tasks to be completed and knows that the final tasks in the goal set require high precision, i.e., requires the laser for object detection. In this case, the user can enter power commands into the keydriver during runtime that forces the laser to power down until the high-precision task is to be executed, at which point the user enters the power commands to bring the laser back up. This example, however, is dependant upon the user's first-hand knowledge and in no way incorporates a software-based power planner.

To verify predictive power planning *without* a user's first hand knowledge of the goal sequence, this thesis makes use of a Markov decision process planner to model the domain. This domain includes a model for high- and low-power consuming sensors as well as tasks that require a high fidelity sensor over a low. The policy graph produced when the MDP plan is solved shows the best action to take at any given state that balances the power consumption with the sensor resolution. There are already open source tools available to solve MDP problems with well-known techniques such as linear programming, value iteration, or policy iteration [17] [55] [64]. In this project, a decision diagram-based toolkit, Stochastic Planning using Decision Diagrams (SPUDD), is utilized [64]. SPUDD solves MDP problems using the value iteration technique on algebraic decision diagrams (ADDs) [7]. ADDs are an extension of binary decision trees that allow for multiple-valued terminal nodes, instead of binary values, which lead to policy graphs depicted as functions of state variables. This creates compact diagrams that are grouped by states with equal variables which are quickly solved [26].

The testing domain is represented in SPUDD through a domain file, included in Appendix A. The MDP representation of the domain includes simulated power consumption for the sensors, sensor accuracy, and tasks that require high or low resolution sensors. First, since the MDP four-tuple, (S, A, P, R), includes an associated reward, $r$, for the selected action taking the system from state, $s$, to $s'$, power consumption is expressed as a negative reward. The policy graph of the MDP maximizes the reward, so the actions will trend toward the lower-cost sensors (without other influences like sensor accuracy, or required fidelity). The sensor accuracy is modeled in the MDP planner by varying the probability of the transition from one state to the next between the laser- and sonar-based actions. Having a higher probability of success causes the actions in the policy graph to trend toward the laser-based actions, provided the reward will offset the cost. Finally, tasks requiring a certain fidelity of the sensor are also modeled through varying the probability in the transition from one state to the next. For example, if the probability of success to move from a low-fidelity state to a high fidelity state when using the sonar is smaller than when using the laser, the policy graph will trend towards actions using the higher resolution sensor, provided the reward is great enough. This process of rewards/costs and probabilities of success provide the balance between power consumption and sensor resolution in the MDP planner.

*3.4.2 Sequencer.* The majority of the resource manager's processing takes place in the sequencer layer. In the initial architecture, the sequencing layer generates behavior hierarchies from the list of goals provided by the deliberator. In order to add power management, the sequencer is broken into two parts, with a top-level Planning and Reasoning module to handle proper information flow from the deliberator, as shown in Figure 3.4. The first part, BehaviorExecutive, has a few minor changes from the initial architecture. Because the BehaviorExecutive is now generating behavior hierarchies in a system that may have limitations placed on its sensor availability, the BehaviorExecutive queries the ResourceManager during behavior hierarchy generation. The resource manager, in turn, provides the availability of each behavior in the hierarchy that is currently being generated based on current sensor statuses. Thus, when the BehaviorExecutive sends an active behavior down to the Controller layer, the behavior will be guaranteed (barring any unforeseen hardware malfunctions) to have the appropriate sensor data available to it since the resource manager was already queried during behavior generation and ensured the sensors are available.

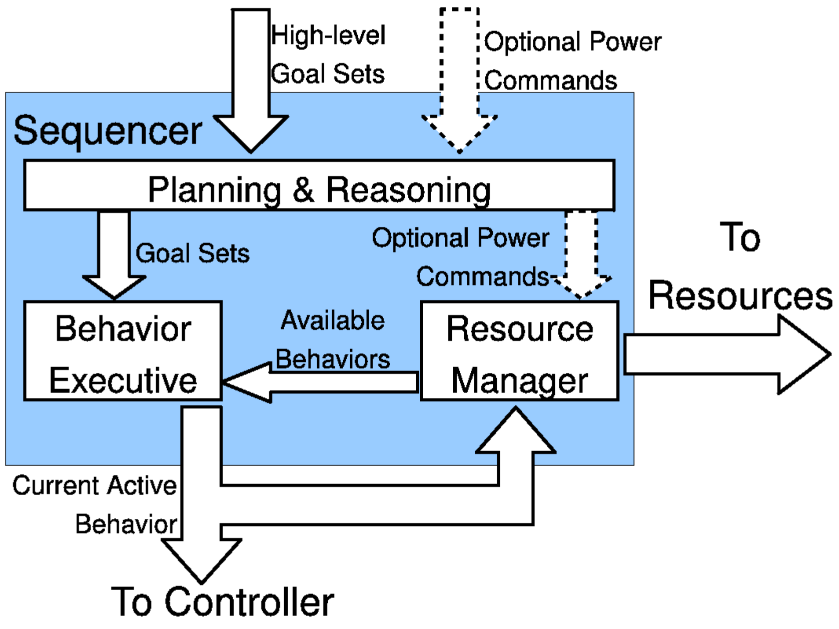


Figure 3.4: Detail of the Sequencer layer. High-level goal sets are received from the Deliberator, goals are processed in the BehaviorExecutive, power commands in the Resource Manager. The Resource Manager passes available behaviors to the BehaviorExecutive which sends the current active behavior to the Controller and Resource Manager.

The second part of the sequencing layer is the resource manager module. The resource manager module handles a few important functions. First, upon system start up and initialization, the resource manager uses the client libraries in Player to make initial connections to all the hardware on the robot. This configures the sensor data input for the robot's internal State reference and allows the sensors to be subscribed or unsubscribed by the resource manager. After successful connection to the sensor device, the internal sensor state is set to AVAILABLE, meaning it is currently subscribed (or powered on) and transferring data. Next, after initialization the resource manager proceeds to an endless loop where it waits to handle hardware changes. If a power requirement is passed down from the deliberator, it will be acted on in this loop. For example, the deliberator can send a request to toggle a particular sensor, such as the laser, on or off. The resource manager then checks to see if the laser is currently subscribed and transmitting data (i.e., the internal status is AVAILABLE) or not subscribed but ACCESSIBLE. If so, the resource manager will unsubscribe from the laser, rendering it inoperable, and change the status to UNAVAILABLE, meaning it is no longer transmitting data and cannot be resubscribed if a subsequent active behavior requires it. If, at the time of receiving a request to activate a sensor, it is already unsubscribed and the status UNAVAILABLE, meaning it will not be resubscribed even if a behavior requires it, then that sensor will be changed to status ACCESSIBLE. Future behaviors could, therefore, use its data. Through this action of toggling devices, the deliberator imparts a form of control on the power usage of the system since it is requiring certain sensors to be up or down, regardless of what the active behaviors require.

The next major function the resource manager performs is managing the current available behaviors. As previously stated, while generating a behavior hierarchy, the BehaviorExecutive queries the resource manager for all available behaviors. During this query, the BehaviorExecutive passes the resource manager a pointer to a set of behaviors. The resource manager steps through the behaviors and checks to see if the required sensors for the particular behavior are AVAILABLE (currently subscribed) or ACCESSIBLE (not currently subscribed, but are allowed to be subscribed). The behaviors with required sensors that are AVAILABLE or ACCESSIBLE then become "usable" sensors with which the BehaviorExecutive can use to construct behavior hierarchies.

Once the BehaviorExecutive has constructed a behavior hierarchy of usable behaviors, it becomes the active set of behaviors. The BehaviorExecutive then sends a message to the resource manager, passing it a representation of the current active behavior set. The resource manager uses this list of required sensors and ensures that all of the corresponding system devices are set to AVAILABLE since the behaviors will need their data. Additionally, the remaining system devices that are not in the list of required sensors are unsubscribed (which can be thought of as "powered down") and the status changed to ACCESSIBLE, since they may be powered back up in a subsequent active behavior set.

The remaining function of the resource manager is to monitor the status of the power supply. During system initialization, if the Player server successfully connects to a power device interface on the robot, the resource manager monitors the actual battery charge levels provided the power source has the capability of software monitoring as detailed in the assumptions (Section 1.4). However, Stage does not include a power interface for its simulated robots. (Actually, Stage does include a power interface, but it does not discharge based on robot usage. It simply always returns 12 volts). Therefore, when connected to Stage, as in during this project's development, a simulated battery is used. The simulated battery is discharged every time the C programming thread is activated. In this implementation, the thread sleeps for one millisecond. Every millisecond, the resource manager polls the system's devices and for each that is currently subscribed (i.e., powered up and running) one milliseconds worth of power usage is discharged from the simulated battery. The simulated battery in this thesis includes discharge amounts for a color pan-tilt-zoom camera (blobfinder), laser range finder, sonar ranger array, mechanical gripper, bumpers, drive motors and onboard microcontroller and computer.

Table 3.1: The amount of power consumed for each device. The first five devices are adjusted by a margin of error to simulate non-linear, real life power consumption. The motor's power is between the two values depending on the robot's current velocity, and the onboard microcontroller and PC is a random amount between the two values.

| **Device** | **Power (watts)** | **Discharge (units/ms)** | **Error** |
|---|---|---|---|
| Laser | 20 | 0.1667 | -20% |
| Blobfinder | 12 | 0.1000 | -20% |
| Gripper | 10 | 0.0833 | -20% |
| Sonar | 0.7 | 0.0058 | +/- 10% |
| Bumpers | 0.25 | 0.0021 | +/- 10% |
| Motors | 0.19–13.29 | 0.0016–0.1108 | – |
| Controller/PC | 12.6–19.6 | 0.1050–0.1633 | – |

The energy discharge amounts were determined through either current research or hardware specifications for products commonly used in robotics applications. Specific hardware discharge amounts are as follows: The blobfinder is modeled after the Sony EVI-D70 pan-tilt-zoom camera system, which has a maximum power consumption of 12 watts [62]. The simulated laser range finder is modeled on the SICK LMS-200 which has an average power consumption of 20 watts [58]. Mei [40] finds that the sonar array in a Pioneer robot consumes approximately 0.7 watts of power. A mechanical servo-gripper sold by Applied Robotics [3] serves as the model for the simulated gripper's 10 watts of power consumption. The bumpers consume virtually negligible amounts of power since they are simply microswitches that wait to be activated. The ten bumpers commonly found on a Pioneer robot consume approximately 25mW each. These devices are then adjusted with a small amount of error that the developer can customize. This is to account for the fact that the discharge amounts are approximations of what would actually be found in real life power consumption. In this project, the error is set either to +/- 10%, which means that the actual amount discharged is a random value inbetween +/- 10% of the specified power consumption, or to -20% for the devices that have a maximum power usage listed, to prevent them from discharging potentially 10% over their specified maximum amount. The drive motors, microcontroller and onboard computer are not subject to the user-specified error. In the case of the drive motors, the power consumption is calculated based on the current velocity of the robot, which Mei [40] shows to be approximately $0.19+13.1v$, where $v$ is the robot's current velocity in meters per second. The onboard microcontroller and computer are discharged as a random value between 12.6 and 19.6 watts, as specified in [40]. All power consumption amounts are summarized in Table 3.1.

The resource manager monitors the battery—either physical or simulated—and causes the system to enter a critical power state once the battery falls below a user-specified threshold. This threshold can be adjusted, depending on the application. Developers may want the robot to function normally all the way until a battery is completely discharged and so may set the threshold at zero (i.e., zero percent of maximum battery capacity, or a dead battery). Initial testing for this project will start with a threshold of 5% initial battery charge, however a range of thresholds are tried. Once the power source falls below this threshold, the robot enters critical power mode. In this mode, the sensors that consume the most power are immediately turned off (for this thesis, unsubscribed) and their internal status is set to UNAVAILABLE so that they will not be resubscribed. The sensors that turn off in critical power mode are also customizable. The developer may want only the laser turned off due to its high power consumption. Another choice is to turn off every device except the sonar array so that the robot will halt its current action upon entering critical power mode and only drive back to a homebase to recharge, using its sonar array to avoid colliding with objects. In testing the behavior-based power management system described here, the laser and blobfinder were turned off in critical power mode since they are the two average highest-power consuming devices at 20 watts and 12 watts, respectively. This also leaves the sonar array operational in case the robot needs to complete a task involving object avoidance, and the gripper in case the robot is in the act of picking an object up or setting it down.

Upon termination of sensors in the critical power mode, a hardware change flag is tripped. This alerts the BehaviorExecutive that it may have to re-plan its hierarchy of behaviors in order to complete the current set of goals. For example, if the current active behavior is to track a certain object and it uses the laser to do so, it will have to re-plan once the laser is unsubscribed in critical power mode. If there is not a replacement behavior in the library that uses a different sensor (e.g., the sonar array) the robot may not be able to fulfill the goals sent to it through the deliberator. It is up to the developer to determine what to do in the situation where a goal cannot be completed. They may decide to skip the goal and move to the next, but often subsequent goals may only be completed once the previous goal is met. Or they may decide to force the robot to travel back to its starting location, although this can be risky if all of its object avoidance behaviors have been eliminated through the termination of sensors in critical power mode. In the testing, the robot simply halts when a goal cannot be reached, because the test cases have temporal constraints that require goals to be completed in order. Temporal constraints are a common occurrence in dynamic real-life environments. Once the BehaviorExecutive does re-plan the behavior hierarchy, it is again passed to the resource manager to ensure the appropriate sensors are still available, even in critical power mode. The current active behavior set is then passed to the controller as normal.

*3.4.3 Controller.* The lowest level in the architecture does not change with the addition of resource management. It still receives the current active behavior set from the BehaviorExecutive in the sequencer. The Unified Behavior Framework in the controller generates a recommended action for each behavior through the use of an arbiter. The recommended action is executed and acted on the motor and device commands which in turn affect the outside world.

The behavior-based resource management system is designed to be flexible and user-programmable to meet a multitude of requirements put forth by each user. It is hoped that use of the system conserves power in a robot, especially in situations where the robot lingers in one spot for long periods, with little movement or sensor interaction. The next sections describe the simulation environment and outline the procedure used in this thesis for testing the power management system.

## *3.5 Design Environment*

The power management system defined in this thesis is designed to function on any mobile robot running this architecture. This may require some extra development prior to execution, but the software architecture has enough inherent modularity that multiple hardware configurations should not be too difficult to implement. During the initial development of the system, it is not feasible to develop, test and configure power management on every possible combination of physical hardware. This makes a software simulation environment particularly useful during the design and development phase. Specifically, this project makes use of the Player/Stage robot simulation environment [65].

The Player project is a device independent client/server package that provides connections to robot hardware through network sockets. It is highly configurable, and allows great flexibility in controller design. The Player server is executed on any machine that has network communication to the robot hardware, and connects to the device clients. Since the control of the clients is handled through network socket commands, any computer programming language that supports socket connections may be used to interface with the robots [25]. The power management system of this thesis makes use of the C programming language for this purpose.

Stage is a two dimensional multi robot simulation environment. Figure 3.5 shows a screenshot of the simulated environment during execution. Stage has animations for most sensors including the target range of the sonars, what the color blobfinder currently has in view, the sweep of the laser range finder and the status of the bumpers. Stage also provides simulated sensors and robotic devices for connection to a Player server. This allows tools to be developed for robot hardware without actually having access to the physical equipment. Stage provides models of the most frequently used robot sensors and devices. Many developers even find that tools and algorithms developed through the connection to Stage require very little modification when connected to the real, physical robot [30] [36] [67]. Thus, the use of the Player

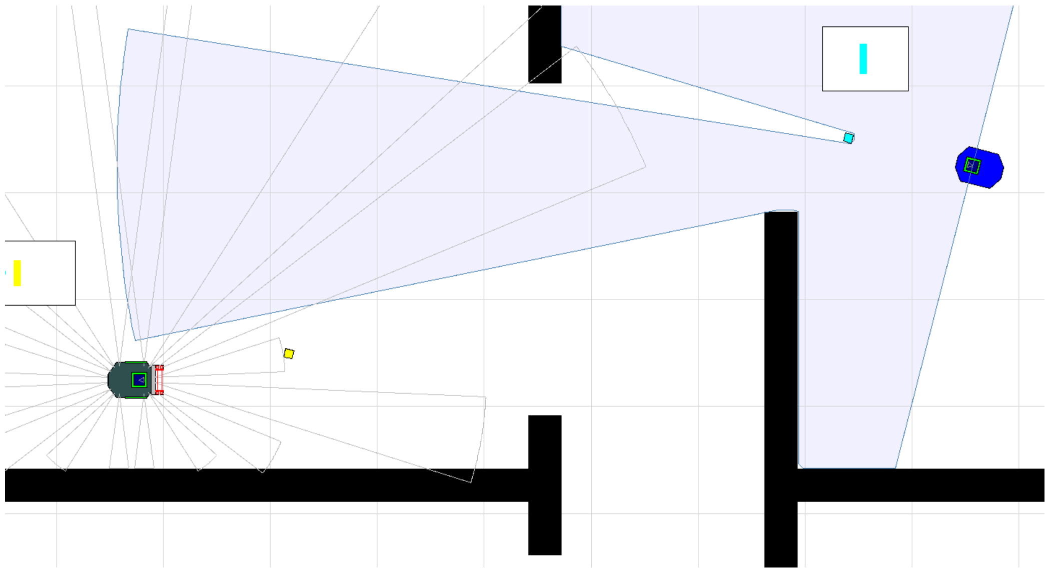


Figure 3.5: Screenshot from Stage simulation environment. The robot on the left is equipped with a gripper and is currently sensing with a sonar array. It detects a block in its blobfinder. The robot on the right does not have a gripper, but is sensing with its 180-degree laser range finder and detects a different block in its blobfinder.

and Stage combination leverages rapid prototyping and development even when access to the required hardware is prevented.

As explained in Section 1.4, the power management system described herein requires access to sensors with the capability of being powered up and down through software control. Additionally, while the proposed power management system will function on a robot (real or simulated) with any classification of sensing hardware, the greatest gains in energy efficiency may be found in robots with many types of sensors. That is, a robot with only an onboard sonar ranging array will not have as high a power savings as that of a robot with a sonar ranging array, a laser range finder, color blobfinder and mechanical gripper.

Therefore, this thesis uses the Player server to control the connections to the Stage robot simulator. This allows the developer to focus on the power management algorithm and not the development of physical hardware that meets the requirements in the Assumptions. Additionally, Stage provides simulations of a greater range of sensing hardware than the developer could readily and physically access. It is in this simulation environment that the design and implementation of the power management system takes place. The details of this architecture are provided in the next section.

## *3.6 Testing Plan*

To ensure the behavior-based resource management's functionality, thorough testing must be performed. As stated in Section 3.5, all testing was accomplished in the Stage simulation environment [65]. This thesis uses a simulated Pioneer P2-AT8 robot [1] which provides the advantages of having hardware and sensor devices readily accessible and configurable, a quick setup and restart time, and easily reproducible testing conditions. The simulation environment required the use of a simulated battery described in Section 3.4.2. Since the battery is simulated and in a simulated environment, testing should not have to run the full 2-3 real-time hours necessary to deplete an average battery in a Pioneer P2-AT8.

There are several situations that must be tested to verify that all aspects of the power management system are fully functional. First, the resource manager must be able to shut down (i.e., unsubscribe) a given sensor when prompted by the deliberator. This also means the resource manager must be able to bring the sensor back up (i.e., resubscribe) if prompted by the deliberator. This ensures the deliberator can exert its control over the power consumption of the robot and possibly allow a measure of predictive power planning for the system. Next, the resource manager must be able to determine the sensors not in use by the current active behavior and turn them off correctly. This is the basis of the behavior-based power management system and as such is quite vital. Finally, the resource manager must be able to force the BehaviorExecutive to re-plan the current behavior hierarchy if a sensor is shut down that was required. For example, when entering critical power mode. This will validate that the resource manager can exert its control over the behaviors that activate.

In order to obtain quantifiable measurements on actual power savings achieved through the use of behavior-based power management, two case studies are considered in the simulation environment previously described. Because it is a simulated environment, the robots are guaranteed to be exact copies of each other. They are, in fact, simulated Pioneer P2-AT8 robots with onboard sonar ranger arrays, laser range finders, color pan-tilt-zoom blobfinders, mechanical grippers, and bumpers. In the first case study, the robots are tested with an active behavior that only wanders around the room in a random pattern. It also provides object avoidance, either through sonar or laser readings. This test is meant to not be very sensor reading-intensive and provide more of a base line of power usage under a light load. The wander behavior is tested on a robot with no power management, a robot with power management that allows any sensor to be used until critical power mode, and a robot with more strict power management that only allows lower-powered sensors to be used unless the behavior specifically requires a higher-powered (and usually higher fidelity) sensor. These three trials should give a good indication of maximum power usage, medium or what is probably the more common power usage, and minimum power usage.

The second case study occurs in the same environment with the same robots, but the robots instead are given a broader task to achieve, i.e., simulated trash collection, instead of the single wander behavior. From their starting location, the robots must travel approximately 15 meters while avoiding obstacles, then wander while searching for a yellow object in the color blobfinder. The object is three meters away and the robot will already be facing towards it. The robot must then move to and pick up the yellow block and drive about 9 meters in the direction of its starting point where it will set the yellow block down. Next the robot will drive 8.5 meters to a new location, then turn in place until a cyan block appears in its color blobfinder. The robot will drive approximately 6 meters to the cyan block and pick it up after which it will proceed another 6 meters away and set the block down. At this point, the robot will start to wander the world randomly while avoiding obstacles until its battery dies. Figure 3.6 depicts the domain in which this case study takes place, including the static obstacles and the objects that will be picked up.

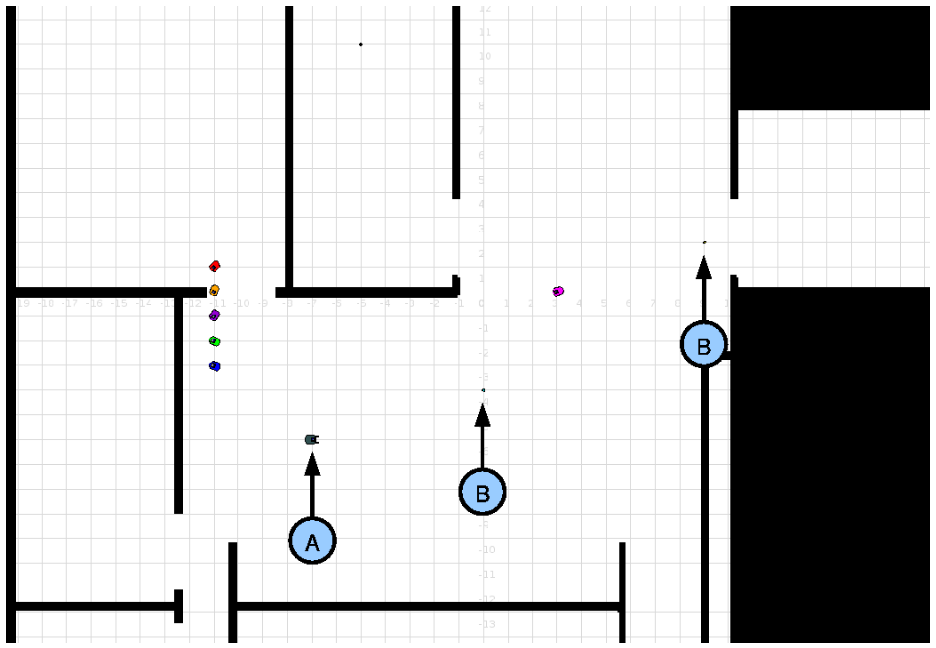


Figure 3.6: A screenshot of the domain in which the robot executes the garbage collection set of goals. The robot starts at A and objects required for pickup are marked B, all other objects and robots are static obstacles.

This domain is modeled in the MDP power planner as a group of six areas—start, zoneA, destA, zoneB, destB, and end—that the robot can travel through. The goal of the robot is to travel to each of the zones, scan for and pick up the block located there, take it to the corresponding destination, and travel to the end location after both blocks have been moved. The robot can travel from one location to the next, as depicted in the directed graph of Figure 3.7. Additionally, zoneB, destB, and the end locations are all simulated as "difficult terrain" in which the laser range finder is much more effective than the sonar. (Instead of difficult terrain, this might also be considered a location where the garbage to pick up is so small, the increased resolution of laser is necessary). This difficulty is represented through the use of probabilities of success. The laser and sonar devices have baseline probabilities of success of 0.9 and 0.8, respectively. This represents the laser's increased accuracy over the sonar in general movement. This increase in accuracy is coupled with an increase in cost, however. Specifically, the laser and sonar are modeled in SPUDD with costs of 4.0 and 1.0, respectively. This represents the extra power needed to use the laser over

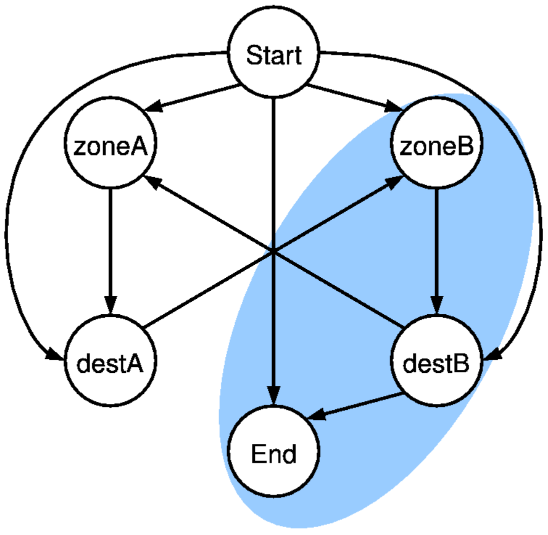


Figure 3.7: Robot location-space as depicted in SPUDD domain. The robot can only move from connected locations in the directions of the arrows. The shaded areas represent "difficult terrain" where the laser range finder is much more effective than the sonar.

the sonar. In the difficult terrain areas, the probabilities of success for the laser and sonar are adjusted to 0.9 and 0.3, respectively. This models the laser range finder as a much better choice to use in difficult areas, even at the higher cost.

This case study is meant to give an indication of a real-life scenario, such as garbage collection, where robots are doing more than just wandering. It is more sensor-intensive than the first case study. This case study is executed on a robot with no power management, a robot with power management that allows any sensor to be used until critical power mode, and a robot with more strict power management that only allows lower-powered sensors to be used unless the behavior specifically requires a higher-powered sensor. Additionally, this case study examines predictive power management in a scenario such as trash collection. Using the Markov decision processed-based planner and SPUDD domain described in Section 3.4.1, two "plans" of sensor usage are developed and executed. The first plan is developed without a power cost for each sensor, while the second plan incorporates the different power costs. The robots using planned power usage are then tested with and without a critical power threshold. Each of these two case studies require a test where the robot has no power management. The following subsection describes how this is possible in the behavior-based power management system.

*3.6.1 No Power Management.* In situations where the developer needs the robot to disable power management features, they need only to change a single value in the source code from true to false. Once power management is disabled, the robot will function with all sensors subscribed (i.e., powered on and transmitting data). Also, the power source, either real or simulated, is no longer monitored to check for the critical power state. Finally, when power management is disabled the robot still responds to commands from the deliberator to toggle sensor devices similar to what is described in Section 3.4.2. However, without power management there is no need for the internal sensor states of AVAILABLE, ACCESSIBLE, and UNAVAILABLE. Therefore, the request from the deliberator to toggle a device simply subscribes (powers up) or unsubscribes (powers down) the device. Without power management enabled, toggling the devices on and off is mostly used for testing and debugging, but the functionality is present.

## *3.7 Summary*

This chapter presented the methodology behind implementing a behavior-based power management system. It was shown that the purpose for such a system is to fill a void in the robotics community that can potentially save onboard battery consumption and thus prolong the untethered life of the robot. The overall design of the system was outlined, followed by a detailed description of the initial architecture that the power management system was constructed on. The environment for simulation was explained along with a brief explanation of the benefits of simulation versus a physical robot. The power management specifications were listed included design for the deliberator, sequencer and controller layers of the architecture. Finally, the chapter concludes with the definition of a testing plan that will show the energy efficiency gains through the use of the behavior-based power management system. The following chapter will present the results of the testing and analyze them for potential insight to the problem statement.

# IV. Results

This chapter presents and analyzes the results obtained through experimentation of the behavior-based power management system. Using the testing plan outlined in Section 3.6, testing and debugging was performed with the goal of verifying system functionality and gathering quantifiable statistics on potential power savings. The first experiment verifies the power management system via proof of concepts. The measurement of potential power savings is determined through the use of two case studies. The first case study executes a low sensor intensity behavior on a robot that wanders randomly around a room. The second case study is a higher sensor intensity set of behaviors that has a robot performing various tasks in an enclosed environment. Results of both case studies are presented at the end of their respective sections. The chapter concludes with a summary of all the findings and overview of total power savings in a robot.

## *4.1 Functionality*

There are three main areas of system functionality that must be verified that they are in working order. As explained in Section 3.6, the power management system must be able to handle device controls when prompted by the deliberator. This means when the deliberator passes down a command, e.g., turn off sonar, the power manager must successfully execute it. This ensures that any power planning that takes place in the deliberator for the purpose of predicting overall power usage can exert control on the resource manager in the sequencer layer.

This concept is verified through the use of the keydriver in the deliberator layer, as described in Section 3.4.1. The keydriver accepts keyboard input from the user of the system. One of the commands that the keydriver has configured is to toggle specific devices on or off. Therefore, as the system is running, one keypress causes the deliberator to issue a command to toggle a sensor, which the resource manager receives. The resource manager then immediately causes the sensor to power off, in the case where it is currently subscribed and transmitting data. This then verifies that the deliberator can exert control over which sensors and devices are powered up or down.

The next piece of system functionality to test is whether the resource manager correctly determines the sensors required for the current active behavior set and power down sensors that are not required. This ability forms the basis of the behavior-based power management system since the goal is to shut down any sensors not currently in use. This functionality is tested through the use of behavior representations as described in Section 3.4.2. Specifically, when the BehaviorExecutive in the sequencer layer has constructed a hierarchy of behaviors that fulfills a goal set, it sends a representation of the current active behavior to the resource manager. The behavior representation includes definitions for goals that each behavior can accomplish, preconditions necessary for execution, postconditions that occur after execution, and most importantly, the sensors required for it generate an action. Using this information, the resource manager steps through all behaviors in the current active behavior set and compiles a list of all the sensors required for execution. After the list is gathered, the resource manager checks the current internal status representations and ensures that all required sensors are either status AVAILABLE or ACCESSIBLE. Recall that in status UNAVAILABLE, the sensor is not to be used for any reason. Using these steps, a known behavior set, such as wander while avoiding objects, can be passed to the resource manager and the resulting sensor statuses can be checked. In this example, it is known that the behavior set, “wander while avoiding objects,” requires a sonar range finder. After the set is passed to the resource manager, the sonar should be AVAILABLE and the remaining sensors are ACCESSIBLE (unless there is mechanical failure or it entered low power mode, in which one or more sensor would be UNAVAILABLE).

The final system functionality check is to test whether the resource manager can force the BehaviorExecutive to plan the current behavior set again because of a hardware change. This is vital once sensors start shutting down e.g., for low power mode. If the current active behavior set requires the laser and the robot enters low power mode where the laser is powered down, the robot has the potential to fail since it is not be able to access the data it needs (i.e., the laser). Ensuring this functionality is slightly more complex. When the BehaviorExecutive gets the signal that a piece of hardware has changed, it requires an alternate plan to be possible. That is, if the BehaviorExecutive makes the plan to use the laser to check for obstacle avoidance and the laser is later shut down, there must be another behavior in the library that the BehaviorExecutive can make use of when replanning. For development of this thesis, the only way these other behaviors were possible was through switching between the laser and the sonar. There were no other devices on the test robot that provided similar input to allow for context switching. For example, the blobfinder could not replace the gripper or the bumper could not replace the laser. However, the laser and sonar are both range finders and can be used in similar manners. The main difference between the two is the laser offers higher fidelity but much higher power consumption over the sonar. Therefore, it was required to develop both sonar-based and laser-based behaviors. Once this is accomplished, sensors can be shut down during execution and the behavior of the robot observed. For example, if the laser is shut down while the robot is wandering and avoiding objects, the robot should seemlessly shut down the laser, switch to the sonar and continue wandering while avoiding obstacles with the possibility of colliding with small objects due to decreased sensor fidelity.

The functionality of the behavior-based power management system is further tested through obtaining quantifiable measurements of power savings. During these experiments, certain parameters are kept constant. These parameters are specified in the following section, after which the two case studies are detailed.

## *4.2 Testing Parameters*

There are several parameters of the power manager that can be adjusted by the developer to suit a specific domain or goal requirement. These are parameters such as the critical power threshold, error in the device power consumption, and actual, measured device power consumption. The following case studies model experiments

Table 4.1: The amount of power consumed for each device. The first five devices are adjusted by a margin of error to simulate non-linear, real life power consumption. The motor's power is between the two values depending on the robot's current velocity, and the onboard microcontroller and PC is a random amount between the two values.

| **Device** | **Power (watts)** | **Discharge (units/ms)** | **Error** |
|---|---|---|---|
| Laser | 20 | 0.1667 | -20% |
| Blobfinder | 12 | 0.1000 | -20% |
| Gripper | 10 | 0.0833 | -20% |
| Sonar | 0.7 | 0.0058 | +/- 10% |
| Bumpers | 0.25 | 0.0021 | +/- 10% |
| Motors | 0.19–13.29 | 0.0016–0.1108 | – |
| Controller/PC | 12.6–19.6 | 0.1050–0.1633 | – |

performed on robots with identical equipment so the actual, measured device power consumption will be kept constant between all robots. These values were stated in Section 3.4.2 and are summarized in Table 4.1. The first five devices are adjusted by an amount of error to simulate power consumption readings that would occur on a real-life sensor. This error amount is developer-specified, and in this project it is kept a constant 10%. Manufacturer's product specifications list the maximum power consumption (not an average) of a device, so in the cases where this value was used, the amount discharged is a random value between -20% and the stated amount whereas the other devices are discharged by a random value between +/- 10% the stated amount.

The developer can also customize at what threshold the robot enters critical power mode. This adjustment potentially has a large effect on robot's actions once the battery has little charge left. If this threshold is set to 0%, the robot will never enter critical power mode and high power consuming sensors are able to completely deplete the battery. However, if the threshold is set to 15%, then the last 15% of battery capacity is spent on lower power consuming devices which could potentially allow the robot to complete additional tasks (albeit with lower fidelity sensors), or return to a recharging station. Therefore, each of the case studies will include trials with the threshold set at varying degrees, from 0% to 15%.

Finally, the developer can choose to what extent critical power mode effects operational sensors. In some scenarios, a low power battery may only trigger the powering down of the laser, for example. In other scenarios it may be prudent to power down every device except the bumpers and rely on "bump and turn" navigation to get the robot to return safely to a location. The sensors that are powered down in critical power mode are highly dependant on the current domain and goal set. In these experiments, the laser and blobfinder will be powered down in critical power mode. These are the two highest power consuming sensors and this choice should return high power savings. The next highest power consuming sensor, the gripper, is left powered on in case the robot is holding an object or about to pick up an object upon entering critical power mode. Leaving the grippers activated lets the robot complete these additional tasks in critical power mode. (Also recall that even though the gripper is left active in critical power mode, its 10W is only consumed if the gripper is in the act of gripping). Lastly, switching off the laser allows the robot to switch to sonar-based behaviors instead and complete additional goals.

## *4.3 Case Study I - Low Sensor Intensity*

This first case study identifies a baseline of power consumption for the system when running three different power configurations. In this study, the load on the sensors is light. That is, not many sensors are required for the overall behavior set, so not much power will be consumed. This scenario runs a behavior set that causes the robot to wander around a room randomly while avoiding contact with objects. An example behavior hierarchy for this task is depicted in Figure 4.1. This hierarchy requires either a laser or a sonar to acquire the ranges to any object around the robot and no other sensors, hence the low sensor intensity heading. This case study is executed on a robot with no power management, a robot with lenient power management where the laser (a high power consuming sensor) is used from the start until low power mode, and a robot with strict power management where the sonar is used from the start since the robot does not require the higher fidelity of a laser. Predictive power

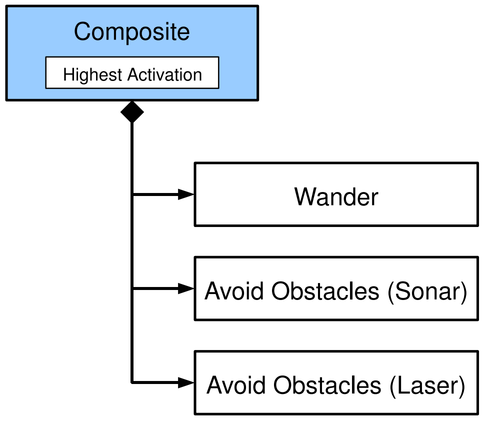


Figure 4.1: An example behavior hierarchy that executes random movement while avoiding obstacles.

planning is not a noteworthy test in this case study, as the robot is only executing one behavior and hence only uses one sensor package. Figure 4.2 shows the trails that result from an example robot wandering in this scenario. All three robots start at the position marked 1 on the map and the trials are run separately. Because these simulations are executed in the Stage environment, there is no power proxy for the Player server to connect to. Therefore, the simulated battery, as described in Section 3.4.2 is monitored. The total initial battery charge is determined somewhat arbitrarily in that it should provide a long enough test to see results, but for practicality should not last 2-3 real time hours. The actual capacity of the battery is irrelevant since all measurements are calculated based on percentage of total charge, however it was set to 500 units. Every one second of real time execution, the system outputs the current battery charge percentage to a text file for later extraction of measurements. Four trials are performed with each of the robots using power management, with their critical power thresholds set to 0%, 5%, 10%, and 15% of initial battery charge. The statistics from all trial runs are examined in the following subsection.

*4.3.1 Results - Case Study I.* It should come as no suprise that the robot *not* using behavior-based power management uses considerably more power that the two

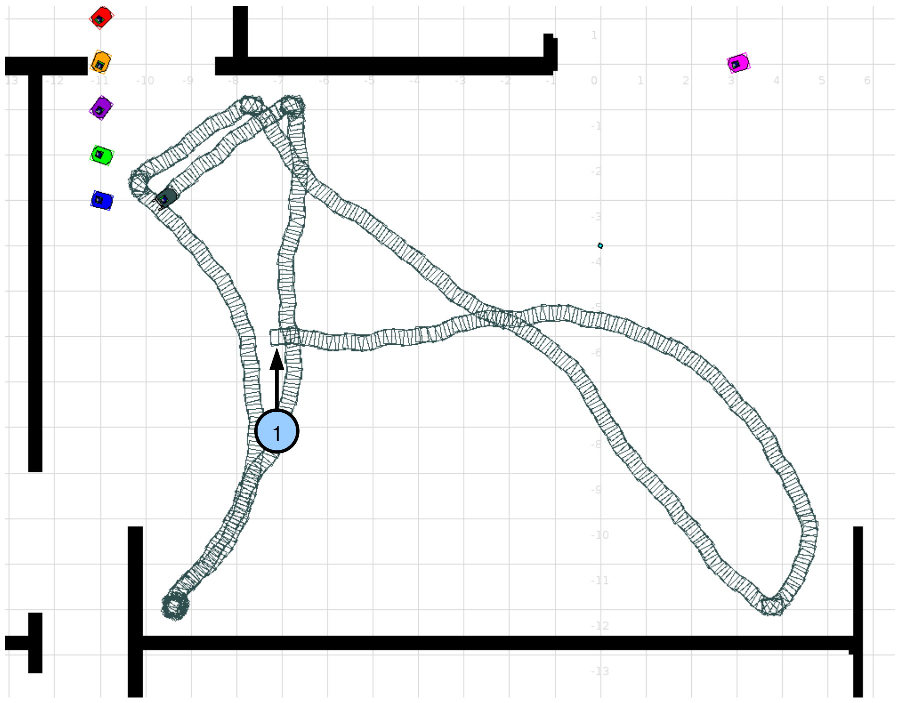


Figure 4.2: A screenshot of the Stage environment showing the robot wandering randomly around the room. The robot starts at 1 and follows the path shown.

that do. This is particularly true in these simulated robots over physical robots since the simulated robots are equipped with lasers, sonars, grippers, bumpers, and color blobfinders all of which consume power while the robot is powered on. A specialized physical robot may be equipped in a similar fashion, but it may be more common to see a robot equipped for a specific purpose and only have one or two sensors burning energy. Figure 4.3 and Table 4.2 show how long each robot lasted before its battery reached zero percent of charge. Notice the robot with no power management had a fully depleted battery in 81 seconds, the robot with lenient power management depleted its battery in 125–140 seconds, depending on the critical power threshold, and the robot with strict power management depleted its battery in an average 232 seconds. Figure 4.4 details the area of the graph where the robots with lenient power management pass into critical power mode. Where low power mode did not activate (i.e., 0% threshold), the battery depleted at 125 seconds. Using critical power mode at 5% of initial battery charge allowed the robot to function an additional 6 seconds

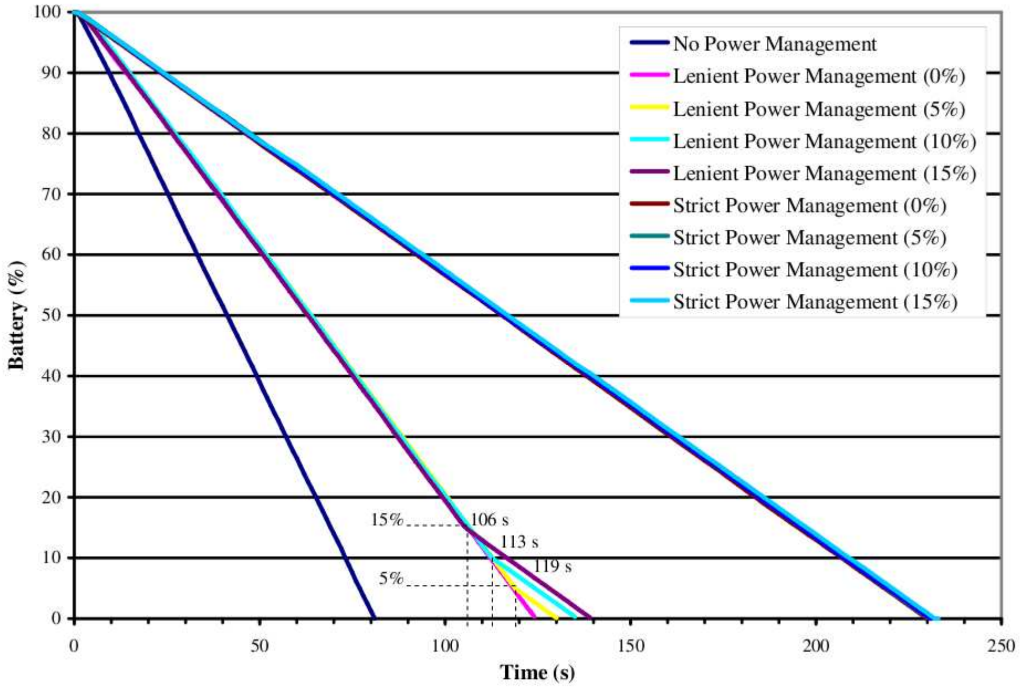


Figure 4.3: The results from three trials of a random wander behavior. The first is with no power management, the next group are trials with power management that uses the laser until critical power mode. The last group uses power management with the sonar throughout.

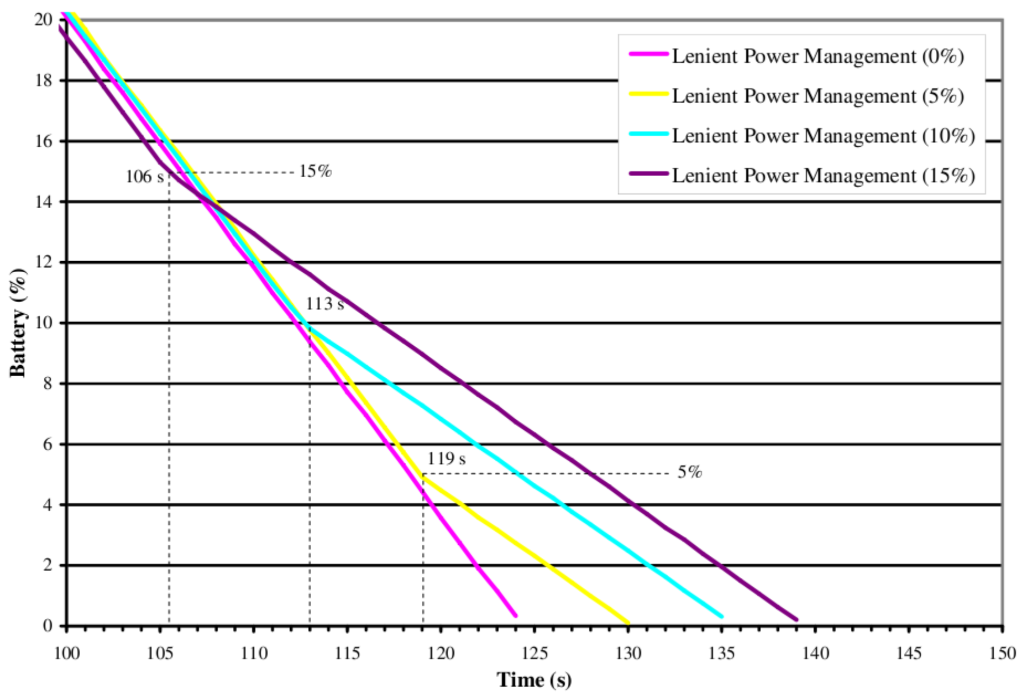


Figure 4.4: The four trials using lenient power management with critical power thresholds set to 0%, 5%, 10%, and 15% and the times they deplete their batteries.

which yielded a 5% increase in the overall lifetime of the battery. Increasing the critical threshold further increased the lifetime of the battery in a linear relationship. This is because the robot is only wandering using the laser to detect objects—no other sensors are running. Once critical power mode is entered, the laser shuts off and the sonars power up, but there are still no other sensors running. Therefore, once in critical power mode—no matter what the threshold—the robots behave exactly like using strict power management. When compared to no power management at all, the robot with lenient power management and a 5% threshold had a 62% increase of time before battery depletion. Finally, the robots with strict power management did not rely on the critical power threshold. They are already restricted in the use of sensors, so critical power mode imposes no further limits. However, strict power management created a 77% increase in lifespan over lenient power management (with 5% threshold) and a huge 186% increase in battery lifetime over no power management. These energy savings are caused by not powering any sensors that are not required by the tested

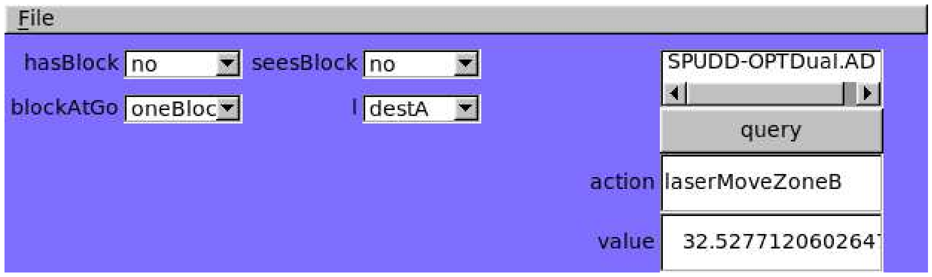
Table 4.2: The amount of time, in seconds, to deplete the battery in each power management mode, with each critical threshold while executing the wander behavior as in Case Study I.

| Threshold | 0% | 5% | 10% | 15% |
|---|---|---|---|---|
| No P.M. | 81 | – | – | – |
| Lenient P.M. | 125 | 131 | 136 | 140 |
| Strict P.M. | 231 | 231 | 231 | 233 |

behavior set, and by using a lower power consuming sensor where possible–in this case, a sonar range finder over a laser.

## *4.4 Case Study II - High Sensor Intensity*

This second case study examines power consumption rates in a scenario more like what might be seen in a real world situation. The load on the sensors is high, since many more sensors will be used at various times while the robots execute a complex set of goals. Figure 4.6 shows the trails that result from an example robot executing this case study. The robot starts at the spot marked 1 and travels approximately 15 meters while avoiding obstacles to the spot marked 2. The robot then wanders while looking for a yellow object in the color blobfinder. The object is at the spot marked 3 and the robot will already be facing towards it. It will pick up the yellow block and drive about 9 meters back towards its starting point where it will set the yellow block down at the position marked 4. Next the robot will drive 8.5 meters through a high-fidelity sensor area to a new location at 5, then turn in place until a cyan block appears in its color blobfinder. The robot will drive approximately 6 meters, again through a high fidelity sensor area, to the cyan block at position 6, and pick it up. The robot then proceeds another 6 meters away through a high-fidelity area and sets the block down at position 7. The robot then begins to wander the world randomly while avoiding obstacles and using a high fidelity sensor until its battery dies. This case study requires the use of a blobfinder, mechanical gripper, and range finder—either laser or sonar. The high fidelity areas are meant to represent either difficult terrain or very small objects to pick up that would require the increased resolution offered



hasBlock no
seesBlock no
blockAtGo oneBloc
l destA
SPUDD-OPTDual.AD
action laserMoveZoneB
value 32.527712060264

Figure 4.5: A screenshot of the SPUDD interface used to trace through a policy graph. Here, it shows to take the laserMoveZoneB action when the robot is currently at DestA with one block at the goal.

by the laser as opposed to the sonar. The sonar might still be able to accomplish the task, but with very low probability.

This case study is again executed on a robot with no power management, a robot with lenient power management where the laser is used from the start until low power mode, and a robot with strict power management where the sonar is used from the start since the robot does not require the higher fidelity of a laser. Each of the power managed robots has four trials, with the critical power thresholds set to 0%, 5%, 10%, and 15%. The varied sensor requirements of this case study allow the predictive power management to also be tested. The domain is modeled and solved using SPUDD for two cases as described in Section 3.4.1 using the definition file in Appendix A. The first case predicts power usage if the power cost of each sensor does not matter, and the second case models the increased power cost of the laser over the sonar. This produces a policy graph of expected rewards/costs for each case that can either be traced through to the specific state and action needed, or it can be queried using SPUDDs user interface as shown in Figure 4.5. Either method of using the policy graph produces the next action for the robot to take, given the input state. During the experiment, the power management system does not communicate directly with SPUDD, so each change in action or behavior is manually entered in the correct sequence. This is possible since the Deliberator's keydriver allows for "on the fly" changes in power requirements, as described in Section 3.4.1.

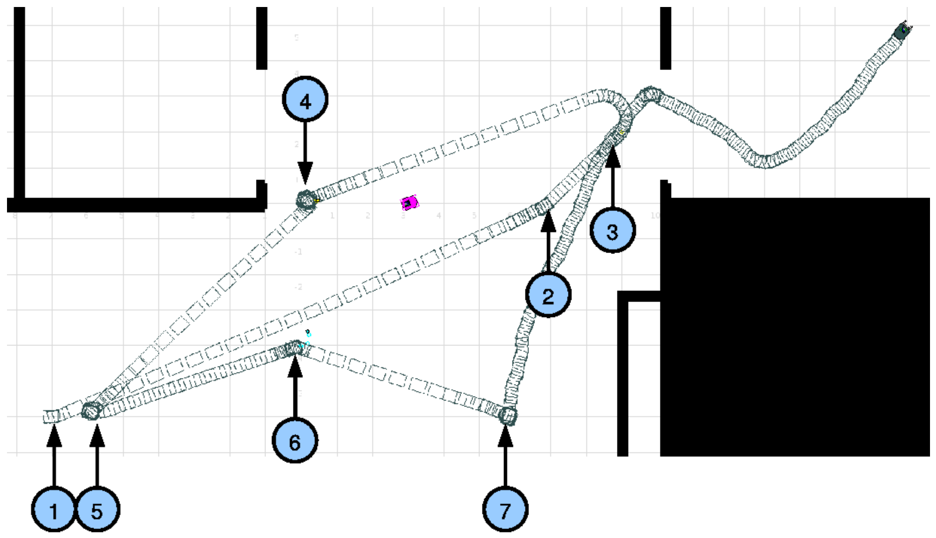


Figure 4.6: A screenshot of the Stage environment showing the robot executing the test plan. The robot starts at 1 and ends at 7 before wandering the domain.

Because each of these simulations are executed in the Stage environment, there is again no power proxy for connection to the Player server. Therefore, the simulated battery, as described in Section 3.4.2 is monitored, just like in Case Study I. Again, the starting capacity is determined somewhat arbitrarily, however comparisons are all percentage-based. The battery's capacity, at 1,000 units, is large enough that the robot with no power management will be able to complete the series of goals and not last for 2–3 real-time hours for convenience. Every one second of real time execution, the system outputs the current battery charge percentage to a text file for later extraction of measurements.

Using the Stage simulation environment also does not allow true stochasticity with the sensors. For example, there is no error on the sonar range readings in Stage, whereas real life sonar sensors tend to be much less accurate. Therefore, the sections of the test domain that are difficult for low-fidelity sensors are considered to be impassable for sonar-based behaviors. That is, the robot simulates bumping into a wall, getting stuck in mud, or some other calamity. The statistics from all trial runs are examined in the following subsection.

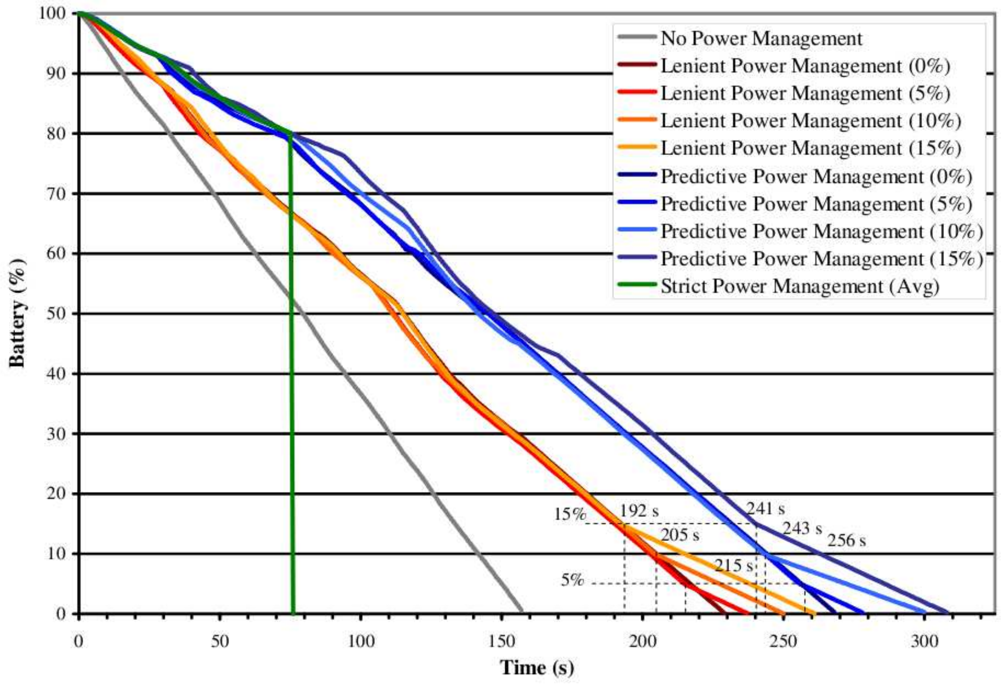


Figure 4.7: The results from four trials of a test sequence of goals. The first is with no power management, the next group is a trial with power management that uses the laser until critical power mode, which is the same as the plan from the predictive power planner when sensor power is not considered. The third group uses predictive power management, including sensor power cost, which forces laser use in the latter half of the test. The last group uses strict power management with the sonar throughout which does not complete the simulation.

*4.4.1 Results - Case Study II.* It should again come as no suprise that the robot *not* using behavior-based power management uses considerably more power than those that do. Figure 4.7 and Table 4.3 show how long each robot lasted before its battery reached zero percent of charge. Notice the robot with no power management had a fully depleted battery in 158 seconds. The robot using a predictive power plan that does not incorporate sensor power cost is exactly the same as a lenient power managed robot. That is, the power plan policy shows the laser as the optimum sensor to use in all situations since it has a higher probability of successful readings over the sonar. These robots that used the laser over the sonar depleted their batteries in 230, 238, 251, and 262 seconds, depending on the critical threshold. The trials with predictive power management including sensor power cost depleted their batteries in 269, 279, 301, and 309 seconds, depending on the critical threshold. The robot using strict power management is simulated to have hit a wall after dropping the first piece of trash since strictly using a sonar in this domain has a high probability of failure after traveling through zone A. Therefore, the robot with strict power management did not finish the simulation.

Figure 4.8 expands the area of the graph where the lenient and predictive power managed robots enter critical power mode. Where the critical threshold was set to 0% (i.e., critical power mode was never activated), the battery is depleted at 230 and 269 seconds for lenient and predictive power management, respectively. Predictive power management therefore provides a 17% increase in battery lifespan. Activating critical power mode at 5% of initial battery charge allowed the robot to function an additional 8 seconds for lenient power management and 10 seconds for predictive power management, which means predictive power management again yields a 17% increase in battery lifetime over lenient. These increases are linear due to the fact at this point in the robot's operation it is only wandering and using the laser (i.e., both plans are the same and no other sensors are powered on). Once critical power mode is activated, the laser is powered off and the sonars are turned on, after which the robot operates just like using strict power management. However, it is noteworthy

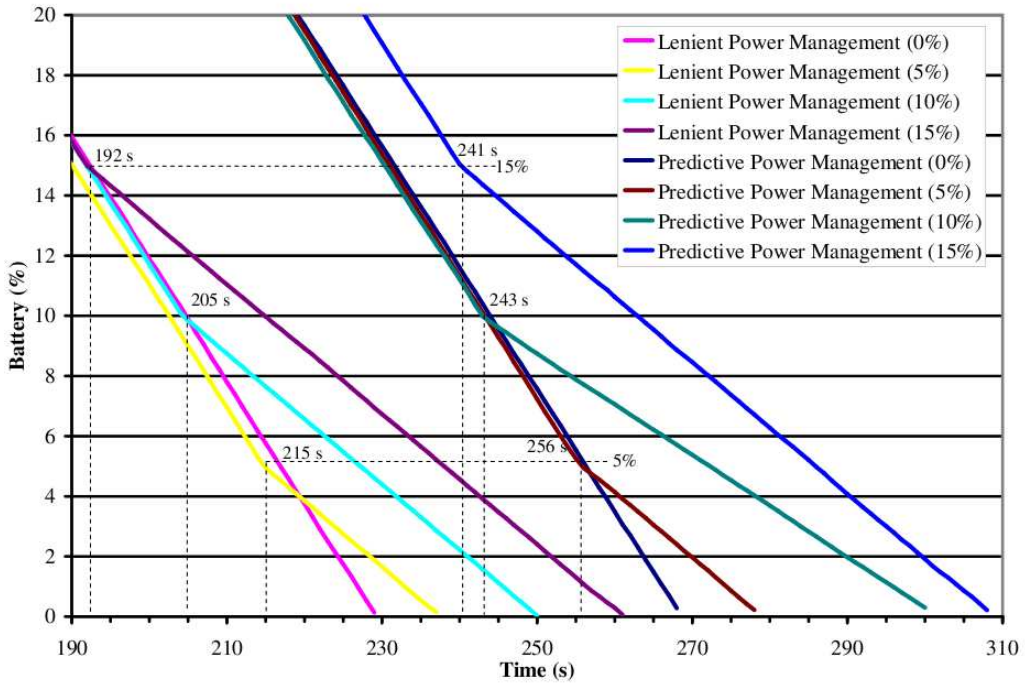


Figure 4.8: The four trials each using lenient and predictive power management with critical power thresholds set to 0%, 5%, 10%, and 15% and the times they deplete their batteries.

that the robot with lenient power management and a 5% critical threshold had a 51% increase in time before battery depletion over no power management, and predictive power management at a 5% threshold was rewarded with a 77% increase in battery lifetime over no power management. Real life scenarios are more likely to encounter situations where high fidelity sensors are required, combined with periods of low fidelity sensor use to conserve battery charge. Therefore, the comparison of lenient power management, where the high power consuming sensors are used, to predictive power management, where low power consuming sensors are used in places where high resolution is not required is the most significant.

Lastly, except for the strict power management, the simulated robots completed the set of goals in approximately 150 seconds. After 150 seconds in Figure 4.7, the robots enter wander mode which creates a power consumption curve that is very linear and not noteworthy. Figure 4.9 shows a close up of the graph where the robots are

Table 4.3: The amount of time, in seconds, to deplete the battery in each power management mode, with each critical threshold while executing a list of goals as in Case Study II.

| Threshold | 0% | 5% | 10% | 15% |
|---|---|---|---|---|
| No P.M. | 158 | – | – | – |
| Lenient P.M. | 230 | 238 | 251 | 262 |
| Predictive P.M. | 269 | 279 | 301 | 309 |
| Strict P.M. | – | – | – | – |

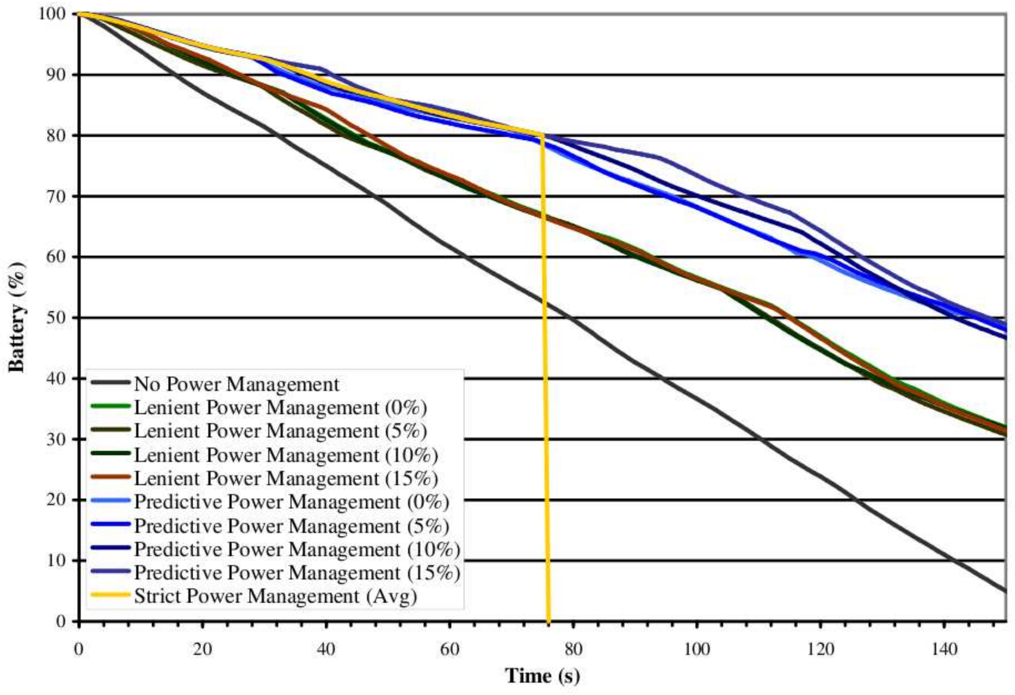


Figure 4.9: A close up of all robots during execution of the set of goals. Except for the strict power management, they complete the task list in approximately 150 seconds, after which the robots wander the domain.

actually executing the list of goals, prior to entering wander mode. This graph shows that using predictive power management yields a 55% increase in remaining battery charge after task completion over lenient power management, and a 860% increase in remaining battery charge over no power management. This is achieved by using the predictive power plan that the MDP power planner produces. In this domain, maximum utility is found by using the sonar in the first collection and the laser in the second. Also shown in Figure 4.9 are the sections of time where greater power is consumed by all robots, especially the high-power consuming set of grippers at 40 seconds and 108 seconds. The following section summarizes the results as a whole and presents some interpretation to their significance.

## *4.5 Summary*

The results from the two case studies show that significant power savings can occur by utilizing behavior-based power management. The greatest savings occur when using strict power management over no power management at all in the low sensor intensity behavior. This makes sense because the robot with power management will keep all its unnecessary sensors turned off, and in this scenario that only requires a range finder, which means all other sensors are off the entire time. The second case study more closely emulates a set of goals that might occur in real-life. Here, the strict power management shows an increase of 168% in the robot battery's lifetime over no power management at all. This number is slightly lower than the first case study, since the complex nature of the required goals cause the robot to use some of its higher-powered sensors. The robot without power management already has these sensors turned on, so the comparative savings drop slightly. These results affirm an important step in allowing robots to last longer when untethered from a static, ground-based power source. The following chapter summarizes the project in its entirety and suggests the next direction for research in this field.

# V. Conclusions

Energy efficiency is of paramount importance in mobile autonomous robots and vehicles. Electronic systems that rely solely on their onboard batteries do best with maximum time out “in the field” before needing to recharge. The longer autonomous vehicles can last before recharging, the more potential they carry for executing tasks. This thesis demonstrated a novel approach to decreasing power consumption in mobile robots and vehicles. This chapter summarizes the project in its entirety first by reiterating the results. Section 5.3 provides vectors for continuing research in this area, followed by the final remarks.

## *5.1 Summary*

The development of the behavior-based power management system fills a void in the robotics world where lack of solid, well-defined energy efficient practises are prevalent. To this point, most power management in robotics was handled by dynamically altering the operating frequency and voltage of the onboard operating system, allowing idle hardware to enter “sleep” mode, or use other techniques built into the software operating system running on the robot [6] [10] [51] [37]. These systems are not inherently built into the control architecture of the robot; rather, they exist in the operating system and device software running on the robot. Besides an efficient way to drive the robot’s motors or plan an efficient path through terrain [40] [38], there has not been a system that focuses on saving power built directly into the underlying robotic architecture.

The behavior-based power management system with predictive power planning is made possible by the development of abstract behavior representations in the sequencer layer [21]. These representations, coupled with their definitions of goals met, preconditions, postconditions, and required sensors, make it possible for the sequencer’s resource manager to take control of all the robot’s hardware (assuming the hardware allows software control, as in Section 1.4). The power management system further ensures that only the sensors currently in use by the active behavior are the only ones actually powered on (Section 4.1). This system, while perhaps simple in concept, has not been executed in any great extent since each traditional behavior-based reactive architecture is tailored for the specific environment or goal the developer has in mind. The system described herein is robust and tailorable enough so that a developer can, with very slight modifications, use this in a wide variety of situations.

This power management system is flexible and also designed to be transparent to other power management schemes that may be in use. It is true, there are other versions of energy efficient algorithms in use. The behavior-based power management system will work seemlessly on top of any that might also be used since the hardware and software behind the system remains unchanged. That is, whatever system may have already been developed on that platform will still function as designed. The power management system described in this project was shown to provide great benefits. These are summarized in the following section.

## *5.2 Results*

Results from testing the behavior-based power management system show that significant power savings are possible. The greatest savings occur when using the lowest power consuming sensors in a low sensor intensity plan, provided the plan does not strictly require sensors of higher power and hence higher fidelity. In situations where the robot wanders randomly in an environment, and does not process great quantities of data with its other sensors, there is great potential for power savings with the behavior-based algorithm described in this thesis. However, even if the robot is required to perform complex goal sets with multiple sensors required at many different times, there is still potential for massive power savings. In fact, this research found up to 96% additional battery life can be had in a robot that uses behavior-based power management. Further, when using a power plan with no sensor power cost, or when strictly using the laser, a 46%–66% increase in battery life is realized over no power management, depending on the threshold of critical power mode. Most importantly, predictive power management that includes the power cost of the sensors, shows an increase in battery lifetime of 70%–96% over no power management at all, depending on the threshold of critical power mode.

These results show significant progress towards longer-lasting battery life in today's robotic endeavors. Robots developed under this architecture have a possibility for approximately 50%–96% longer times away from their human handlers and out in the world. This provides developers with robots that have significantly longer loiter times. For example, if the robots are used to setup a wireless sensor network, they can travel out to their node's destination and remain on station for a significantly longer period of time on a single battery charge before they must be either replaced or travel back to the base station for recharging. Other applications for future work are described in the next section.

## *5.3 Future Work*

While behavior-based power management provides a great step in longer-lasting batteries on robots, there is always room for more savings given that a battery is by its nature a finite device. One such place for improvement is making the deliberator more efficient. As the deliberator, or planner, has global knowledge of the tasks to be performed by the robot, it could have a sense of the overall power consumption for the robot, more so than the Markov decision process modeled in this thesis. This is where a high-efficiency, real-time predictive power planner could be utilized. The deliberator can perform cost-benefit analysis of the current set of goals and determine the most efficient way to balance the power requirements with the fidelity requirements of the system, during execution.

Similarly, this power management system will have to be performed in real time. The only way to guarantee real time is through a real time operating system. Executing behavior-based power management in a real time operating environment will be met with its own unique set of problems and challenges. However, certain situations, like those of critical infrastructure or high availability require the use of true real time systems. If these types of systems are present in robotics, they will also need maximum energy efficiency to be effective.

Finally, the behavior-based power management system could go one step farther and adjust which sensors are currently powered up based on the operating frequency of the hardware as well as the current active behavior. For example, if a robot is wandering and avoiding objects, it will need a range finder, either laser or sonar. However, if that robot is moving at a very slow crawl it will not need to use the ranger constantly. Perhaps only one reading per second would be enough. Similarly, if the robot is moving at 1m/s it might need the constant readings that a laser or sonar provides. However, if the robot is in a mostly empty environment, it may be able to further decrease the rate of sensing in order to make the battery last just enough longer to reach a recharging station, for example.

## *5.4 Final Remarks*

Autonomous mobile robots are becoming more and more prevalent in various parts of the industry. NASA frequently sends robots to explore our astral neighbors. Law enforcement often uses robots to examine suspicious packages where it would be risky to send a human. The military is exploring uses for robots on the battlefield by detecting mines, carrying casualties off the front lines, searching for chemical weapons or performing forward reconnaissance. Robots are frequently placed into situations where a human life would be put in danger. The longer a robot can last in these situations before a battery needs recharging means the more humans that *don't* have to risk their lives. This thesis provides one small step in that direction by introducing a new approach to more energy efficient robots.

```
////////////////////////////////////////////////////////////////////
// Trash Collecting Robot
//
// robot travels to different areas (A and B)
// picks up the block in each area,
// takes the block to the proper destination in each area
// then ends in a final area (wandering)
// Robot has some freedom of choice for which area to go to next.
// Blocks can be "scanned" and "picked up" inside Zones.
// Blocks are dropped in Dest's.
// The laser is more reliable to go from one area to the next
// Only certain locations can be traveled to from others:
// start -> all locations ZoneA -> DestA
// DestA -> ZoneB ZoneB -> DestB
// DestB -> ZoneA, End End -> End
// The laser is much more reliable in "difficult areas"
// Difficult areas are: ZoneB, DestB, End
/////////////////////////////////////////////////////////////////////


(variables
   // hasBlock: which block is the robot holding?
   (hasBlock blockA blockB no)
   // seesBlock: which block can the robot see?
   (seesBlock blockA blockB no)
   // blockAtGoal: which block is at its goal?
   (blockAtGoal oneBlock twoBlocks none)
   (l start zoneA destA zoneB destB end)   // l: the robot's location
)
```

```
// do nothing - not sure why you'd want to do this
action nothing
   hasBlock (SAMEhasBlock)
   seesBlock (SAMEseesBlock)
   blockAtGoal (SAMEblockAtGoal)
   l (SAMEl)
endaction

///////////////////////////////////////
// All the laser-based move actions
///////////////////////////////////////
action laserMoveZoneA
   hasBlock (SAMEhasBlock)
   seesBlock (SAMEseesBlock)
   blockAtGoal (SAMEblockAtGoal)
   //laser is more accurate,
   //so probability is higher of reaching the next area
   l (l (start (0.1 0.9 0.0 0.0 0.0 0.0))
    (zoneA (0.0 1.0 0.0 0.0 0.0 0.0))
    (destA (0.0 0.0 1.0 0.0 0.0 0.0))
       (zoneB (0.0 0.0 0.0 1.0 0.0 0.0))
    (destB (0.0 0.9 0.0 0.0 0.1 0.0))
    (end (0.0 0.0 0.0 0.0 0.0 1.0)))
   cost (4.0)
endaction

action laserMoveDestA
   hasBlock (SAMEhasBlock)
```

```
    seesBlock (SAMEseesBlock)
    blockAtGoal (SAMEblockAtGoal)
    //laser is more accurate,
    //so probability is higher of reaching the next area
    l (l (start (0.1 0.0 0.9 0.0 0.0 0.0))
     (zoneA (0.0 0.1 0.9 0.0 0.0 0.0))
     (destA (0.0 0.0 1.0 0.0 0.0 0.0))
     (zoneB (0.0 0.0 0.0 1.0 0.0 0.0))
     (destB (0.0 0.0 0.0 0.0 1.0 0.0))
     (end (0.0 0.0 0.0 0.0 0.0 1.0)))
    cost (4.0)
endaction

action laserMoveZoneB
    hasBlock (SAMEhasBlock)
    seesBlock (SAMEseesBlock)
    blockAtGoal (SAMEblockAtGoal)
    //laser is more accurate,
    //so probability is higher of reaching the next area
    l (l (start (0.1 0.0 0.0 0.9 0.0 0.0))
     (zoneA (0.0 1.0 0.0 0.0 0.0 0.0))
     (destA (0.0 0.0 0.1 0.9 0.0 0.0))
     (zoneB (0.0 0.0 0.0 1.0 0.0 0.0))
     (destB (0.0 0.0 0.0 0.0 1.0 0.0))
     (end (0.0 0.0 0.0 0.0 0.0 1.0)))
    cost (4.0)
endaction

action laserMoveDestB
```

```
   hasBlock (SAMEhasBlock)
   seesBlock (SAMEseesBlock)
   blockAtGoal (SAMEblockAtGoal)
   //laser is more accurate,
   //so probability is higher of reaching the next area
   l (l (start (0.1 0.0 0.0 0.0 0.9 0.0))
    (zoneA (0.0 1.0 0.0 0.0 0.0 0.0))
    (destA (0.0 0.0 1.0 0.0 0.0 0.0))
    (zoneB (0.0 0.0 0.0 0.1 0.9 0.0))
    (destB (0.0 0.0 0.0 0.0 1.0 0.0))
    (end (0.0 0.0 0.0 0.0 0.0 1.0)))
   cost (4.0)
endaction

action laserMoveEnd
   hasBlock (SAMEhasBlock)
   seesBlock (SAMEseesBlock)
   blockAtGoal (SAMEblockAtGoal)
   //laser is more accurate,
   //so probability is higher of reaching the next area
   l (l (start (0.1 0.0 0.0 0.0 0.0 0.9))
    (zoneA (0.0 1.0 0.0 0.0 0.0 0.0))
    (destA (0.0 0.0 1.0 0.0 0.0 0.0))
    (zoneB (0.0 0.0 0.0 1.0 0.0 0.0))
    (destB (0.0 0.0 0.0 0.0 0.1 0.9))
    (end (0.0 0.0 0.0 0.0 0.0 1.0)))
   cost (4.0)
endaction
```

```
////////////////////////////////
// sonar-based move actions
////////////////////////////////
action sonarMoveZoneA
   hasBlock (SAMEhasBlock)
   seesBlock (SAMEseesBlock)
   blockAtGoal (SAMEblockAtGoal)
   //sonar not very accurate,
   //so larger chance of not making it to next area
   l (l (start (0.2 0.8 0.0 0.0 0.0 0.0))
    (zoneA (0.0 1.0 0.0 0.0 0.0 0.0))
    (destA (0.0 0.0 1.0 0.0 0.0 0.0))
    (zoneB (0.0 0.0 0.0 1.0 0.0 0.0))
    (destB (0.0 0.8 0.0 0.0 0.2 0.0))
    (end (0.0 0.0 0.0 0.0 0.0 1.0)))
   cost (1.0)
endaction

action sonarMoveDestA
   hasBlock (SAMEhasBlock)
   seesBlock (SAMEseesBlock)
   blockAtGoal (SAMEblockAtGoal)
   //sonar not very accurate,
   //so larger chance of not making it to next area
   l (l (start (0.2 0.0 0.8 0.0 0.0 0.0))
    (zoneA (0.0 0.2 0.8 0.0 0.0 0.0))
    (destA (0.0 0.0 1.0 0.0 0.0 0.0))
    (zoneB (0.0 0.0 0.0 1.0 0.0 0.0))
    (destB (0.0 0.0 0.0 0.0 1.0 0.0))
```

```
     (end (0.0 0.0 0.0 0.0 0.0 1.0)))
    cost (1.0)
endaction

action sonarMoveZoneB
    hasBlock (SAMEhasBlock)
    seesBlock (SAMEseesBlock)
    blockAtGoal (SAMEblockAtGoal)
    //sonar not very accurate,
    //so larger chance of not making it to next area
    1 (1 (start (0.7 0.0 0.0 0.3 0.0 0.0))
     (zoneA (0.0 1.0 0.0 0.0 0.0 0.0))
     (destA (0.0 0.0 0.7 0.3 0.0 0.0))
     (zoneB (0.0 0.0 0.0 1.0 0.0 0.0))
     (destB (0.0 0.0 0.0 0.0 1.0 0.0))
     (end (0.0 0.0 0.0 0.0 0.0 1.0)))
    cost (1.0)
endaction

action sonarMoveDestB
    hasBlock (SAMEhasBlock)
    seesBlock (SAMEseesBlock)
    blockAtGoal (SAMEblockAtGoal)
    //sonar not very accurate,
    //so larger chance of not making it to next area
    1 (1 (start (0.7 0.0 0.0 0.0 0.3 0.0))
     (zoneA (0.0 1.0 0.0 0.0 0.0 0.0))
     (destA (0.0 0.0 1.0 0.0 0.0 0.0))
     (zoneB (0.0 0.0 0.0 0.7 0.3 0.0))
     (destB (0.0 0.0 0.0 0.0 1.0 0.0))
     (end (0.0 0.0 0.0 0.0 0.0 1.0)))
    cost (1.0)
 endaction

 action sonarMoveEnd
    hasBlock (SAMEhasBlock)
    seesBlock (SAMEseesBlock)
    blockAtGoal (SAMEblockAtGoal)
    //sonar not very accurate,
    //so larger chance of not making it to next area
    l (l (start (0.7 0.0 0.0 0.0 0.0 0.3))
     (zoneA (0.0 1.0 0.0 0.0 0.0 0.0))
     (destA (0.0 0.0 1.0 0.0 0.0 0.0))
     (zoneB (0.0 0.0 0.0 1.0 0.0 0.0))
     (destB (0.0 0.0 0.0 0.0 0.7 0.3))
     (end (0.0 0.0 0.0 0.0 0.0 1.0)))
    cost (1.0)
 endaction

 /////////////////////////////////////////////
 //scan for a block - only works in zones
 //////////////////////////////////////////////
 action scan
    hasBlock (SAMEhasBlock)
    seesBlock (seesBlock (blockA (l (start (0.0 0.0 1.0))
     (zoneA (1.0 0.0 0.0))
     (destA (0.0 0.0 1.0))
     (zoneB (0.0 1.0 0.0))
     (destB (0.0 0.0 1.0))
     (end (0.0 0.0 1.0))))
     (blockB (l (start (0.0 0.0 1.0))
     (zoneA (1.0 0.0 0.0))
     (destA (0.0 0.0 1.0))
     (zoneB (0.0 1.0 0.0))
     (destB (0.0 0.0 1.0))
     (end (0.0 0.0 1.0))))
     (no (l (start (0.0 0.0 1.0))
     (zoneA (1.0 0.0 0.0))
     (destA (0.0 0.0 1.0))
     (zoneB (0.0 1.0 0.0))
     (destB (0.0 0.0 1.0))
     (end (0.0 0.0 1.0)))))
   blockAtGoal (SAMEblockAtGoal)
   l (SAMEl) //don't move while scanning
endaction

/////////////////////////////////////////////
//pickup the block that it currently scans
/////////////////////////////////////////////
action pickup
   hasBlock (hasBlock (blockA (seesBlock (blockA (1.0 0.0 0.0))
     (blockB (0.0 1.0 0.0))
     (no (0.0 0.0 1.0))))
     (blockB (seesBlock (blockA (1.0 0.0 0.0))
     (blockB (0.0 1.0 0.0))
     (no (0.0 0.0 1.0))))
     (no (seesBlock (blockA (1.0 0.0 0.0))
    (blockB (0.0 1.0 0.0))
    (no (0.0 0.0 1.0)))))
   seesBlock (SAMEseesBlock) //keep picked up block in viewfinder
   blockAtGoal (SAMEblockAtGoal)
   l (SAMEl)
endaction

//////////////////////////////////////////////////
//drop block, update which blocks are at goals
// only works at dest's
//////////////////////////////////////////////////
action drop
   //only able to drop blocks in certain locations
   hasBlock (hasBlock (blockA  (l (start (1.0 0.0 0.0))
    (zoneA (1.0 0.0 0.0))
    (destA (0.0 0.0 1.0))
    (zoneB (1.0 0.0 0.0))
    (destB (1.0 0.0 0.0))
    (end (1.0 0.0 0.0))))
    (blockB  (l (start (0.0 1.0 0.0))
    (zoneA (0.0 1.0 0.0))
    (destA (0.0 1.0 0.0))
    (zoneB (0.0 1.0 0.0))
    (destB (0.0 0.0 1.0))
    (end (0.0 1.0 0.0))))
    (no  (0.0 0.0 1.0)))
   //no longer see block in viewfinder
   seesBlock (seesBlock (blockA (0.0 0.0 1.0))
    (blockB (0.0 0.0 1.0))
```

```
 (no  (0.0 0.0 1.0)))
//which block at the goal is dependent upon which is in the gripper
blockAtGoal (blockAtGoal (oneBlock (hasBlock
                     (blockA (l (start (0.0 0.0 1.0))
 (zoneA (0.0 0.0 1.0))
 (destA (1.0 0.0 0.0))
 (zoneB (0.0 0.0 1.0))
 (destB (0.0 0.0 1.0))
 (end (0.0 0.0 1.0))))
 (blockB (l (start (0.0 0.0 1.0))
 (zoneA (0.0 0.0 1.0))
 (destA (0.0 0.0 1.0))
 (zoneB (0.0 0.0 1.0))
 (destB (0.0 1.0 0.0))
 (end (0.0 0.0 1.0))))
 (no (0.0 0.0 1.0))))
 (twoBlocks (hasBlock (blockA (0.0 1.0 0.0))
 (blockB (0.0 1.0 0.0))
 (no (0.0 0.0 1.0))))
 (none (hasBlock (blockA (l (start (0.0 0.0 1.0))
 (zoneA (0.0 0.0 1.0))
 (destA (1.0 0.0 0.0))
 (zoneB (0.0 0.0 1.0))
 (destB (0.0 0.0 1.0))
 (end (0.0 0.0 1.0))))
 (blockB (l (start (0.0 0.0 1.0))
 (zoneA (0.0 0.0 1.0))
 (destA (0.0 0.0 1.0))
 (zoneB (0.0 0.0 1.0))
```

```
    (destB (1.0 0.0 0.0))
    (end (0.0 0.0 1.0))))
    (no (0.0 0.0 1.0)))))
   l (SAMEl)
endaction

//overall reward function
reward [+   (l (start (0.0))
    (zoneA (0.0))
    (destA (0.0))
    (zoneB (0.0))
    (destB (0.0))
    (end  (2.0)))
    (blockAtGoal (oneBlock (1.0))
    (twoBlocks (5.0))
    (none (0.0)))]
discount 0.900000
tolerance 0.1
```

# *Bibliography*

1. ActivMedia Robotics, 2007. URL `http://www.activrobots.com`.
2. Akyildiz, IF, W. Su, Y. Sankarasubramaniam, and E. Cayirci. "Wireless sensor networks: a survey". *Computer Networks*, 38(4):393–422, 2002.
3. Applied Robotics, 2007. URL `http://www.arobotics.com/index.cfm`.
4. Austin, T., E. Larson, and D. Ernst. "SimpleScalar: an infrastructure for computer system modeling". *Computer*, 35(2):59–67, 2002.
5. Aydin, H., R. Melhem, D. Mosse, and P.M. Alvarez. "Dynamic and Aggressive Scheduling Techniques for Power-Aware Real-Time Systems". *Proceedings of IEEE Real-Time Systems Symposium*, 95–105, 2001.
6. Aydin, H., R. Melhem, D. Mosse, and P. Mejia-Alvarez. "Power-aware Scheduling for Periodic Real-Time Tasks". *Computers, IEEE Transactions on*, 53(5):584–600, 2004.
7. Bahar, RI, EA Frohm, CM Gaona, GD Hachtel, E. Macii, A. Pardo, and F. Somenzi. "Algebraic Decision Diagrams and Their Applications". *IEEE/ACM International Conference on CAD*, 188–191, 1993.
8. Balch, T. and R.C. Arkin. *Motor Schema-based Formation Control for Multiagent Robot Teams*. Technical report, Georgia Institute of Technology, 1994.
9. Barrett, A. and D.S. Weld. "Partial-order planning: evaluating possible efficiency gains". *Artificial Intelligence*, 67(1):71–112, 1994.
10. Benini, L., A. Bogliolo, and G. De Micheli. "A Survey of Design Techniques for System-Level Dynamic Power Management". *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 8(3):299–316, 2000.
11. Benini, L., A. Bogliolo, GA Paleologo, and G. De Micheli. "Policy Optimization for Dynamic Power Management". *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 18(6):813–833, 1999.
12. Boutilier, C., T. Dean, and S. Hanks. "Decision-theoretic planning: Structural assumptions and computational leverage". *Journal of Artificial Intelligence*, 11:1–94, 1999.
13. Brateman, J., C. Xian, and Y.H. Lu. "Energy-Effcient Scheduling for Autonomous Mobile Robots". *Very Large Scale Integration, 2006 IFIP International Conference on*, 361–366, 2006.
14. Brooks, D., V. Tiwari, and M. Martonosi. "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations". *Proceedings of the 27th Annual International Symposium on Computer Architecture*, 83–94, 2000.

15. Brooks, R. “A robust layered control system for a mobile robot”. *Robotics and Automation, IEEE Journal of [legacy, pre-1988]*, 2(1):14–23, 1986.

16. Burger, D. and T.M. Austin. “The SimpleScalar tool set, version 2.0”. *ACM SIGARCH Computer Architecture News*, 25(3):13–25, 1997.

17. Cassandra, A. R. “POMDP Page, The”, 2005. URL `http://www.cassandra.org/pomdp/code/index.shtml`.

18. Connell, JH. “A behavior-based arm controller”. *Robotics and Automation, IEEE Transactions on*, 5(6):784–791, 1989.

19. Doherty, L., K.S.J. Pister, and L. El Ghaoui. “Convex Position Estimation in Wireless Sensor Networks”. *Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies, Proceedings of IEEE*, 3:1655–1663, 2001.

20. D’Souza, C., B.H. Kim, and R. Voyles. “Morphing Bus: A rapid deployment computing architecture for high performance, resource-constrained robots”. *Robotics and Automation, 2007 IEEE International Conference on*, 311–316, 2007.

21. Duffy, J. *Dynamic Behavior Sequencing in a Hybrid Robot Architecture*. Master’s thesis, Air Force Institute of Technology, 2008.

22. Estrin, D., L. Girod, G. Pottie, and M. Srivastava. “Instrumenting the world with wireless sensor networks”. *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP’01). 2001 IEEE International Conference on*, 4, 2001.

23. Firby, R.J. *Adaptive execution in complex dynamic worlds*. Ph.D. thesis, Yale University, 1989.

24. Gat, E. et al. “On three-layer architectures”. *Artificial Intelligence and Mobile Robots*, 195–210, 1998.

25. Gerkey, B., R. Vaughan, and A. Howard. “The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems”. *Advanced Robotics, Proceedings of International Conference on*, 2003.

26. Hoey, J., R. St-Aubin, A. Hu, and C. Boutilier. “SPUDD: Stochastic planning using decision diagrams”. *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, 279–288, 1999.

27. Hui, J., Z. Ren, and B.H. Krogh. “Sentry-Based Power Management in Wireless Sensor Networks”. *The 2nd International Workshop on Information Processing in Sensor Networks (IPSN03)*, 2003.

28. Huntsberger, TL, G. Rodriguez, and P.S. Schenker. “Robotics challenges for robotic and human mars exploration”. *Proceedings of ROBOTICS2000*, 299–305, 2000.

29. iRobot, 2007. URL `http://www.irobot.com/`.

30. Jung, B. and G.S. Sukhatme. "Tracking Targets Using Multiple Robots: The Effect of Environment Occlusion". *Autonomous Robots*, 13(3):191–205, 2002.

31. Kaelbling, L.P. *An Architecture for Intelligent Reactive Systems*. Center for the Study of Language and Information, 1987.

32. Kim, W., D. Shin, H.S. Yun, J. Kim, and S.L. Min. "Performance comparison of dynamic voltage scaling algorithms for hard real-time systems". *Real-Time and Embedded Technology and Applications Symposium, 2002. Proceedings. Eighth IEEE*, 219–228, 2002.

33. Littman, M.L., T.L. Dean, and L.P. Kaelbling. "On the complexity of solving Markov decision problems". *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, 394–402, 1995.

34. Maes, P. "The dynamics of action selection". *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 2:991–997, 1989.

35. Mainwaring, A., D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. "Wireless sensor networks for habitat monitoring". *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, 88–97, 2002.

36. Makarenko, A.A., S.B. Williams, F. Bourgault, and H.F. Durrant-Whyte. "An experiment in integrated exploration". *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2002.

37. Marinoni, M., T. Facchinetti, G. Buttazzo, and G. Franchino. "An Embedded Real-Time System for Autonomous Flight Control". *Proceedings of the 2006 International Congress on Methodologies for Emerging Technologies in Automation*, 2006.

38. Mei, Y. *Energy-Efficient Mobile Robots*. Ph.D. thesis, Purdue University, 2006.

39. Mei, Y., Y.H. Lu, YC Hu, and CSG Lee. "Energy-efficient motion planning for mobile robots". *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, 5, 2004.

40. Mei, Y., Y.H. Lu, YC Hu, and CSG Lee. "A Case Study of Mobile Robot's Energy Consumption and Conservation Techniques". *Advanced Robotics, 2005. ICAR'05. Proceedings., 12th International Conference on*, 492–497, 2005.

41. Mei, Y., Y.H. Lu, Y.C. Hu, and C.S.G. Lee. "Deployment of Mobile Robots with Energy and Timing Constraints". *Robotics, IEEE Transactions on [see also Robotics and Automation, IEEE Transactions on]*, 22(3):507–522, 2006.

42. Mei, Y., C. Xian, S. Das, Y.C. Hu, and Y.H. Lu. "Repairing Sensor Network Using Mobile Robots". *Proc. of the ICDCS Int. Workshop on Wireless Ad hoc and Sensor Networks, Lisboa, Portugal*, 2006.

43. Mei, Y., C. Xian, S. Das, Y.C. Hu, and Y.H. Lu. "Replacing Failed Sensor Nodes by Mobile Robots". *Workshop on Wireless Ad Hoc and Sensor Networks*, 2006.

44. Mei, Y., C. Xian, S. Das, Y.C. Hu, and Y.H. Lu. “Sensor replacement using mobile robots”. *Computer Communications*, 30(13):2615–2626, 2007.

45. Mejia-Alvarez, P., E. Levner, and D. Mossé. “Adaptive scheduling server for power-aware real-time tasks”. *ACM Transactions on Embedded Computing Systems (TECS)*, 3(2):284–306, 2004.

46. Min, R., M. Bhardwaj, S.H. Cho, A. Sinha, E. Shih, A. Wang, and A. Chandrakasan. “Low-Power Wireless Sensor Networks”. *VLSI Design*, 3–7, 2001.

47. Moravec, H.P. *Robot Rover Visual Navigation*. UMI Research Press Ann Arbor, Mich, 1981.

48. Murphy, R.R. *Introduction to AI robotics Intelligent robots and autonomous agents*. The MIT Press, 2000.

49. Nguyen, H.G. and J.P. Bott. “Robotics for Law Enforcement: Applications Beyond Explosive Ordnance Disposal”. *SPIE Proc. 4232: Technologies for Law Enforcement*, 5–8, 2000.

50. Nilsson, N.J. “Shakey the Robot”. *DTIC Research Report ADA458918*, 1984.

51. Pillai, P. and K.G. Shin. “Real-time dynamic voltage scaling for low-power embedded operating systems”. *Proceedings of the eighteenth ACM symposium on Operating systems principles*, 89–102, 2001.

52. Puterman, M.L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc. New York, NY, USA, 1994.

53. Ratnakumar, BV, MC Smart, RC Ewell, LD Whitcanack, KB Chin, and S. Surampudi. “Lithium-Ion rechargeable batteries on Mars Rover”. *2nd International Energry Conversion Engineering Conference, Providence, Rhode Island, August 15-18, 2004*, 2004.

54. Rosenblatt, J. “Utility Fusion: Map-Based Planning in a Behavior-Based System”. *Field and Service Robotics*, 411–418, 1998.

55. Russell, S.J. and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1995.

56. Shih, E., S.H. Cho, N. Ickes, R. Min, A. Sinha, A. Wang, and A. Chandrakasan. “Physical Layer Driven Protocol and Algorithm Design for Energy-Efficient Wireless Sensor Networks”. *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, 272–287, 2001.

57. Shin, Y. and K. Choi. “Power Conscious Fixed Priority Scheduling for Hard Real-Time Systems”. *Proceedings of the 36th ACM/IEEE Conference on Design Automation*, 134–139, 1999.

58. SICK - Sensor Intelligence, 2007. URL `http://www.sick.com`.

59. SimpleScalar, LLC, 2004. URL `http://www.simplescalar.com`.

60. Simunic, T., P. Glynn, and G. De Micheli. “Dynamic power management for portable systems”. *Proceedings of the 6th annual international conference on Mobile computing and networking*, 11–19, 2000.

61. Sinha, A., A. Chandrakasan, and C. MIT. “Dynamic Power Management in Wireless Sensor Networks”. *Design & Test of Computers, IEEE*, 18(2):62–74, 2001.

62. Sony - Broadcast and Business Solutions Company, 2007. URL `http://bssc.sel.sony.com/BroadcastandBusiness/index.shtml`.

63. Sony Aibo Europe, 2007. URL `http://support.sony-europe.com`.

64. SPUDD, Welcome to, 2007. URL `http://www.computing.dundee.ac.uk/staff/jessehoey/spudd/index.html`.

65. The Player Project, 2007. URL `http://playerstage.sourceforge.net/`.

66. Tiwari, V., D. Singh, S. Rajgopal, G. Mehta, R. Patel, and F. Baez. “Reducing power in high-performance microprocessors”. *Proceedings of the 35th annual conference on Design automation-Volume 00*, 732–737, 1998.

67. Vaughan, RT, K. Stoy, GS Sukhatme, and MJ Mataric. “LOST: localization-space trails for robot teams”. *Robotics and Automation, IEEE Transactions on*, 18(5):796–812, 2002.

68. Woolley, B. *Unified Behavior Framework for Reactive Robot Control in Real-Time Systems*. Master’s thesis, Air Force Institute of Technology, 2007.

69. Xian, C., Y.H. Lu, and Z. Li. “Energy-aware scheduling for real-time multiprocessor systems with uncertain task execution time”. *Proceedings of the 44th annual conference on Design automation*, 664–669, 2007.

70. Yamauchi, B.M. “PackBot: a versatile platform for military robotics”. *Proceedings of SPIE*, 5422:228–237, 2004.

71. Ye, W., J. Heidemann, and D. Estrin. “Medium Access Control with Coordinated Adaptive Sleeping for Wireless Sensor Networks”. *Networking, IEEE/ACM Transactions on*, 12(3):493–506, 2004.

72. Zheng, R. and R. Kravets. “On-demand Power Management for Ad Hoc Networks”. *Ad Hoc Networks*, 3(1):51–68, 2005.

| REPORT DOCUMENTATION PAGE | *Form Approved* *OMB No. 0704-0188* |
|---|---|

| | | |
|---|---|---|
| **1. REPORT DATE** *(DD-MM-YYYY)* 27-03-2008 | **2. REPORT TYPE** MASTER'S THESIS | **3. DATES COVERED** *(From - To)* AUG 2006 - MAR 2008 |

| | |
|---|---|
| **4. TITLE AND SUBTITLE** BEHAVIOR-BASED POWER MANAGEMENT IN AUTONOMOUS MOBILE ROBOTS | **5a. CONTRACT NUMBER** |
| | **5b. GRANT NUMBER** |
| | **5c. PROGRAM ELEMENT NUMBER** |
| **6. AUTHOR(S)** FETZEK, CHARLES A, FIRST LIEUTENANT, USAF | **5d. PROJECT NUMBER** JON 07-107/07-138 |
| | **5e. TASK NUMBER** |
| | **5f. WORK UNIT NUMBER** |
| **7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)** AIR FORCE INSTITUTE OF TECHNOLOGY<br>GRADUATE SCHOOL OF ENGINEERING AND MANAGEMENT (AFIT/EN)<br>2950 HOBSON WAY, BLDG 640<br>WRIGHT-PATTERSON AFB, OH 45433-8865 | **8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCE/ENG/08-05 |
| **9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)** DR. JACOB CAMPBELL, AFRL/RYRN<br>2241 AVIONICS CIRCLE, BLDG 620<br>WRIGHT-PATTERSON AFB, OH 45433-7335<br>(293) 255-6127 x4154<br>jacob.campbell@wpafb.af.mil | **10. SPONSOR/MONITOR'S ACRONYM(S)** |
| | **11. SPONSOR/MONITOR'S REPORT NUMBER(S)** |

**12. DISTRIBUTION/AVAILABILITY STATEMENT**
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

**13. SUPPLEMENTARY NOTES**

**14. ABSTRACT**
Current attempts to prolong a robot's battery life focus on outdated techniques that have high overhead and are not built in to the underlying robotic architecture. In this thesis, battery life is extended through development of a behavior-based power management system, including a Markov decision process (MDP) power planner. This system examines sensors needed by the currently active behavior set and powers down those not required. Predictive power planning models the domain as an MDP problem in the Deliberator. The planner creates a power policy that accounts for current and future power requirements in stochastic domains. This provides a power plan that uses lower-power consuming devices at the start of a goal sequence in order to save power for the areas where higher-power consuming sensors are needed. Power savings are observed in two case studies: Low and high sensor intensity environments. Testing reveals that in a real life scenario involving multiple goals and multiple sensors, the robot's battery charge can be extended up to 96% longer when using this system over robots that rely on traditional power management.

**15. SUBJECT TERMS**
Dynamic Power Management; Robotic Architecture; Energy Savings; Markov Decision Process; Predictive Power Planning

| 16. SECURITY CLASSIFICATION OF: | | | 17. LIMITATION OF ABSTRACT | 18. NUMBER OF PAGES | 19a. NAME OF RESPONSIBLE PERSON |
|---|---|---|---|---|---|
| a. REPORT | b. ABSTRACT | c. THIS PAGE | | | DR. GILBERT L. PETERSON |
| U | U | U | UU | 98 | **19b. TELEPHONE NUMBER** *(Include area code)* (937) 785-3636, gilbert.peterson@afit.edu |

Reset **Standard Form 298** (Rev. 8/98)
Prescribed by ANSI Std. Z39.18