3-17-2009

Unified Behavior Framework in an Embedded Robot Controller

Stephen S. Lin

Unified Behavior Framework in an Embedded Robot Controller

THESIS

Stephen S. Lin, Captain, USAF

AFIT/GCE/ENG/09-04

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCE/ENG/09-04

Unified Behavior Framework in an Embedded Robot Controller

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

Stephen S. Lin, B.S.E.E.

Captain, USAF

March 2009

AFIT/GCE/ENG/09-04

Unified Behavior Framework in an Embedded Robot Controller

Stephen S. Lin, B.S.E.E.
Captain, USAF

Approved:

[illegible]	17 MAR 09
Dr. Gilbert Peterson, PhD (Chairman)	date
[illegible]	17 MAR 09
Lt Col Michael Veth, PhD (Member)	date
Barry Mullins	17 Mar 09
Dr. Barry Mullins, PhD (Member)	date

AFIT/GCE/ENG/09-04

Abstract

Robots of varying autonomy have been used to take the place of humans in dangerous tasks. While robots are considered more expendable than human beings, they are complex to develop and expensive to replace if lost. Recent technological advances produce small, inexpensive hardware platforms that are powerful enough to match robots from just a few years ago. There are many types of autonomous control architecture that can be used to control these hardware platforms. One in particular, the Unified Behavior Framework, is a flexible, responsive control architecture that is designed to simplify the control system's design process through behavior module reuse, and provides a means to speed software development. However, it has not been applied on embedded systems in robots. This thesis presents a development of the Unified Behavior Framework on the Mini-$WHEGS^{TM}$, a biologically inspired, embedded robotic platform. The Mini-$WHEGS^{TM}$ is a small robot that utilize wheel-legs to emulate cockroach walking patterns. Wheel-legs combine wheels and legs for high mobility without the complex control system required for legs. A color camera and a rotary encoder completes the robot, enabling the Mini-$WHEGS^{TM}$ to identify color objects and track its position. A hardware abstraction layer designed for the Mini-$WHEGS^{TM}$ in this configuration decouples the control system from the hardware and provide the interface between the software and the hardware. The result is a highly mobile embedded robot system capable of exchanging behavior modules with much larger robots while requiring little or no change to the modules.

Acknowledgements

To my adviser who pushed me to get things done.

To my fellow students with whom I shared the journey.

And to my wife without whom I'm completely helpless.

Stephen S. Lin

Table of Contents

Page

List of Figures

Figure Page

List of Abbreviations

Unified Behavior Framework in an Embedded Robot Controller

I. Introduction

Robots have been used to take on dangerous tasks for many years under the direct control of human operators. Where teleoperation is impractical, autonomous control systems carry on while having limited contact with the operators. The dangerous or remote nature of the tasks also require the autonomous robots to be robust enough to survive the accomplishment of the tasks. Yet these requirements often drive development cost to such levels that few robots can be acquired and that the most dangerous of tasks must be abandoned to ensure the robot's survival. Another issue is the size of robust, autonomous robots which limit the operating environment to large, open spaces.

Recent technological advances allow autonomous control systems to operate on small, inexpensive hardware platforms. Besides opening a new realm of tasks for autonomous robots that human operators have difficulty accomplishing, inexpensive robots are far more expendable. Groups of less robust, yet expendable, robots can deploy to accomplish the sort of task that the previous generation of high-cost robots are not risked to perform. Being smaller, they can also operate in confined spaces where even human operators cannot reach.

The keys to creating small, inexpensive, autonomous robots is the control system that makes it autonomous and operability in its target environment. This control system must be responsive to be useful in a real environment, flexible enough to perform different tasks when required, and be usable on the variety of possible specialized hardware platforms to keep the development cost low. And since the target environments are small, enclosed spaces with uneven surfaces, the physical form of the robot cannot simply be a scaled down version of the large robots that can only operate on level ground. The combination of these requirements drive the development of a new embedded, autonomous robot, and mark the beginning of a new generation of highly mobile, low cost, autonomous systems.

1.1 Research Goal

The most intuitive development path for a small robot that operates in small, enclosed spaces, is to model the robot after creatures normally found there. For a responsive, highly mobile robot, insects are the ideal model. The first objective of this research is to develop the embedded robot controller mounted in a small robot as a viable, flexible hardware platform for a general purpose autonomous robot. This research adapts the proven Unified Behavior Framework (UBF) [25] to the limited resources of an embedded controller. The Unified Behavior Framework brings the benefit of simplifying development, code reuse, scalability, and choice of behavior system for the robot. There have been other autonomous embedded controller robots but none whose control architecture exhibits such properties. Second, the specific robotic platform to be used, the Mini-$WHEGS^{TM}$ [15], has never been made fully autonomous. This platform utilizes the unique properties of wheels and legs to cross rough terrain. While wheels are very simple to use in locomotion, they only perform well on flat, open areas. Its opposite, the leg, is able to traverse uneven terrain just as well as flat, level ground, but require a complex control system for each leg that may dominate the computational resources of an embedded processor. The combination of the Unified Behavior Framework and a legged hardware platform makes a insectoid creature that can be programmed to perform a wide variety of tasks.

1.2 Sponsor

This research is sponsored by the Intelligent Navigation, Sensing, and Cooperative Tasking (INSeCT) for the Air Force Office of Scientific Research (AFOSR). INSeCT is located at the Precision Navigation and Time division of the Air Force Research Laboratories (AFRL/RYR) at Wright-Patterson Air Force Base. INSeCT requires small, autonomous robots for operations in confined spaces and as low cost fleets. The work presented in this thesis provides a solution that is compatible with continuing work on larger robots and paves the way to cooperative development between the embedded and larger robots.

1.3 Assumptions

Although the techniques and methods presented in this thesis apply to any object oriented language, C/C++ is natively supported by the embedded Linux operating system and is the language of choice. The Unified Behavior Framework used in this research is a non-real-time version of the original development [25] and is written in C/C++. Basic knowledge of C/C++ and objected oriented concepts are assumed when discussing the UBF.

1.4 Thesis Organization

This thesis is divided into five chapters. This chapter introduces the problem and the goals of the research. Chapter II presents an overview of several types of autonomous control architectures and discuss the advantages and disadvantages of each compared to the Unified Behavior Framework. Chapter II also presents a number of embedded and biologically-inspired robots, highlighting the advantageous qualities of $WHEGS^{TM}$ locomotion. Chapter III outlines the development of the robot, from the individual hardware components to the extensions of UBF which adapt it to the embedded platform. This is followed by the results of developing the hardware platform and the operation of UBF executing a demonstration behavior on the Mini-$WHEGS^{TM}$. Finally, Chapter V summarizes the lessons learned, discusses areas for future research encountered during the development process are discussed.

II. Autonomous Architectures and Robots Background

Just as numerous inventors of the past look to birds to find inspiration for flying machines, robot designers look to nature for existing forms that perform the functions they require. When the goal is for a machine to take the place of a human in a dangerous situation, designers copied all parts of the human required to do the job. For scurrying about confined spaces, exploring and searching for targets of interest with possibly the additional goal of remaining undetected, the insect is the inspiration of choice. Although other biological organisms also exhibit the required characteristics, insects combine flexible locomotion and simpler mechanical form.

This chapter presents an overview of several types of robot control architectures and a spectrum of small, autonomous robot projects as well as developments in embedded solutions suitable for small, autonomous robots. This chapter introduces recent architectures, comparing them to reactive behavioral architectures, in particular, the Unified Behavioral Framework. These are followed by an examination of several biologically-inspired robots, and autonomy that has been added to these platforms, with emphasis on the Mini-$WHEGS^{TM}$ which is derived from the cockroach.

2.1 Sense-Plan-Act Paradigm

The Sense-Plan-Act (SPA) architecture is similar to building a computer program [10]. A human programmer collects specifications, writes, and executes the program. Similarly, the SPA architecture divides the task into three functional units: Sense gathers information about the environment, Plan devises a set of actions, Act executes the actions. Figure 2.1a shows a graphical representation of the architecture.

The most time consuming and complex component of the SPA architecture is the maintenance of the internal state that represents the sensed world [25]. The next two steps of SPA depend on this internal state exclusively so it must be as accurate as possible. Also, because these two steps depend on the internal world model, the sensing step must be completed before the planning step can begin. In the planning

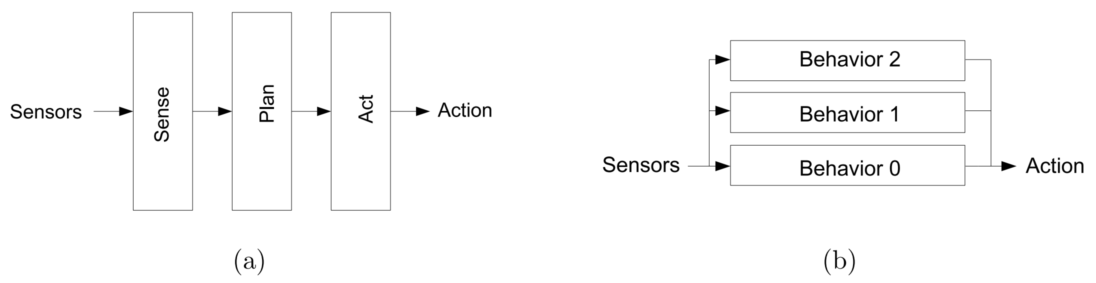


Figure 2.1: (a) Sense-Plan-Act Paradigm. (b) Reactive Behavior Paradigm.

step, the complete plan of action is formulated to reach the goals. Using the complete internal representation of the world, the planning state plots each intermediate step required to reach the goal state from the current state. Finally, the plan is carried out in the final stage which interfaces directly with the physical hardware on the robot. After each action, the sensing state activates again to update the internal world model and restart the Sense-Plan-Act cycle.

The SPA architecture was first demonstrated in Shakey the Robot [17]. However, it also shows a serious limitation of SPA. Planning and world modeling are computationally very intensive. The result is that in the sensing stage when the internal world model is being constructed, there is no plan ready for execution and thus no action to express. After the the sensing state completes and while the planning stage is active, the robot is unresponsive to the changing environment. The result is that the robot is incapable of dealing with highly dynamic environments

Other concerns to note are the open and closed world assumption and the frame problem [16]. Using the closed world assumption means the internal world model contains everything the robot needs to know about its environment. The model must contain all conceivable details about the robot's operating environment but it is also very easy to miss details. Robots programmed to operate on the closed world assumption can fail if it encounters anything unexpected in its environment. With the open world assumption, the system is designed to be flexible enough to handle such unexpected events. The frame problem is the attempt to limit the size of the robot's local environment so the resulting world model is workable. Instead of wasting computation time on objects and events that will not affect the robot in the immediate future, concentrating on the local environment greatly reduces the computational requirements of forming the world model. However, the required size of the local environment also depend on the goals of the robot and the nature of the environment.

2.2 Reactive Paradigm

In the early 1980's, two very similar responses to the issues in SPA appeared from Braitenberg [5] and Brooks [6]. Braitenberg presented a series of biologically-based thought vehicles that were configurations of sensors, motors, and interconnections that give behavioral responses to stimuli. By combining very simple mechanisms in such a way that relatively complex behavior is produced, he avoids over-designing a behavior to reach the same level of complexity. On the other hand, the resulting behavior of any single configuration is very difficult to predict since that behavior directly linked to environmental stimuli.

From the same start point of behaviors that emulate simple organisms, Brooks explores a robot architecture built using simple behaviors that operate purely on sensing and acting. Other designs follow the same theme: minimize the use of a time consuming internal state to minimize the delay between sensing and acting. This type of design, diagrammed in Figure 2.1b for comparison with SPA, is call the Reactive Paradigm.

2.2.1 Subsumption. Brooks' subsumption architecture [6] decompose the functional units vertically, focusing on the resulting external behavior. In SPA the functional units are decomposed horizontally, which leads to a time-consuming chain of modules that must execute in sequence. The vertical decomposition of subsumption creates levels of competence, which are classes of behavior for the robot over all environments. A higher level of competence is a more capable behavior. In this way, each layer is one complete, functional, control system where the more traditional function units of SPA cannot work independently of each other. Also, a level of competence subsumes the levels below it to produce the final behavior. This system also allow the multiple layers to work toward different goals. The issue of integrating multiple sensors to generate a state transforms into an issue of integrating multiple behaviors resulting from those sensor inputs. Since the lower levels are functional at a level of competence, if the more "abstract" higher behavior has trouble producing a result, a sensible behavior is still produced, making this a robust system that is responsive to a changing environment. Finally, additional sensors/behaviors can easily be added. Each layer executes independently of all the other layers so to add a sensor or to add a layer of competence to a system with a fully utilized processor is possible by simply running the new layer on an additional processor. The required amount of communications between layers is low so the complexity in coordinating multiple processor is minimized.

2.2.2 Potential Field. Another type of reactive architecture generates potential fields to guide the robot. Potential fields consists of vectors that point away from obstacles or toward a goal. If fully generated, this is a complete plan from any point in the robot's environment to the goal.

Arkin's motor schema [3] approach makes use of potential fields in place of layers of behavior modules that subsume each other. These motor schemas take sensory inputs to produce a motor command. All commands to the same motor are summed and normalized to produce the final motor command. Only the vector at the current location is generated. This system produces the vector for the point on the potential field the robot occupies to eliminate the need to have knowledge of anything other then what the sensors are detecting at the moment. If the robot is initialized at random locations and the motor command vectors are recorded, a complete potential field forms.

Payton adds internal state and a certain amount of planning back to the basic reactive architecture [19]. All knowledge and constraints relevant to the goal forms an internalized plan which is pre-generated, stored, and updated as needed to account for changes in the environment. This internalized plan consists of a gradient field that is similar to a potential field. Payton utilizes the gradient field as an additional input to a subsumption architecture so it remains responsive to a dynamic environment but retain the ability to have a centralized goal and storage past experiences.

2.2.3 Unified Behavior Framework. Most reactive control systems are designed and customized for each use. This leaves the robot tied to the strengths and weaknesses of the reactive architecture that its control system is based on. The behavior modules within the control system are also tied to each other, the controller that binds them together, and the underlying hardware. This makes behavior module reuse difficult and necessitates a new reactive control system for each platform. The Unified Behavior Framework (UBF) [25] is a reactive architecture designed to overcome the shortcomings of such specially constructed reactive control systems to create a readily reusable reactive architecture.

The UBF uses object oriented programming (OOP) concepts to create a generic framework to integrate behavior modules. The main issues with monolithic control systems are that they are tied to the platform and that their components are tied to each other. A generic state object provides a generic interface to sensor data and other state information from the platform and a generic action object provides a generic interface to the motors and any other actions the platform is capable of. These two objects provide the common interface for behavior modules to be reusable in any UBF based control system. Each behavior module is derived from a generic behavior object that specifies the generation of an action object. This allows the reactive controller to select behaviors at runtime without needing to customize the behavior module to the controller. The result of encapsulating these components of a control system is an architecture that encourages reuse of behavior modules that are usable on any platform.

There is also a construct that encapsulate multiple behaviors that derives from the behavior object. The composite object is a set of behavior modules with a run-time selectable arbiter object to reduce the set of action objects to one action. This allows complex behaviors to be built out of simpler, independently developed behaviors. Since each composite behavior can be used in the place of any ordinary behavior object, any arbitrary hierarchy of composite objects and behaviors modules are possible, allowing any reactive architecture to be built and included within or alongside of each other.

2.3 Message Based

A property that is not often considered is the extendability of the architecture, both in hardware and in software. OpenR [9], developed to control entertainment-oriented robots, focuses on the interfaces between components and linkages between components. Using OpenR objects and a system of inter-object communications [4], the architecture allows plug-and-play capabilities for hardware and software components. These OpenR objects each execute in parallel and pass messages to each other to “see” through the sensor objects and act through motor control objects. Network and hierarchies of interconnected objects allow higher level behaviors. A limitation is the dependence on message passing bandwidth. Large numbers of objects or just several camera objects that need to pass large amounts of data to other objects for processing can overwhelm the internal communications bandwidth. A greater problem stems from behaviors that are linear combinations of component behaviors. Each component cannot start processing until the previous component in the chain completes processing and pass along the required data and the combined behavior cannot be produced until the final component of the chain completes processing. The latency of the chained behavior may be too long for the robot to be responsive.

2.4 Hybrid

While SPA architectures are too slow to respond to changing environments, reactive architectures sacrifice long term planning and goal-seeking for responsiveness. Hybrid architectures seek to combine the best features of both paradigms by including a planning module that does not interfere with the reactive elements of the architecture.

2.4.1 Three-Layered Architecture. Gat's three-layer architecture [10] is a variant of the hybrid architecture that adds a module, the sequencer, between the slow, deliberative planner and the fast, reactive controller. This is diagrammed in Figure 2.2. The controller is tightly coupled to the sensors and actuators and responds immediately to any stimuli. It contains a library of primitive behaviors that require little or no need for state information to keep it responsive to the real world instead of the last state update. The sequencer then activates primitive behaviors as needed to carry out a plan. The sequencer also responds to any unexpected situation it may encounter while the plan is being carried out. Another constraint on the sequencer is time. Whatever algorithm is implemented as part of the sequencer cannot take a long time relative to the rate of environmental change. This generally means search algorithms and certain vision processing must be completed at the deliberator level. The deliberator is the least constrained layer but it is probably also the least invoke layer since all the time consuming algorithms end up there. The deliberator can be called upon to generate a plan or to respond to requests from the sequencer. These three layers are easily separable from each other, allow very different implementations in each layer, and make unambiguous divisions.

2.4.2 Saphira. The Saphira architecture [12] is centered around the internal Local Perceptual Space (LPS) and a version of the Procedural Reasoning System (PRS). It is comparable to the three-layered architecture with the LPS and PRS in the sequencing level which can query the planner for path planning and control the set of basic behaviors in the reactive layer. The goal of the architecture is to

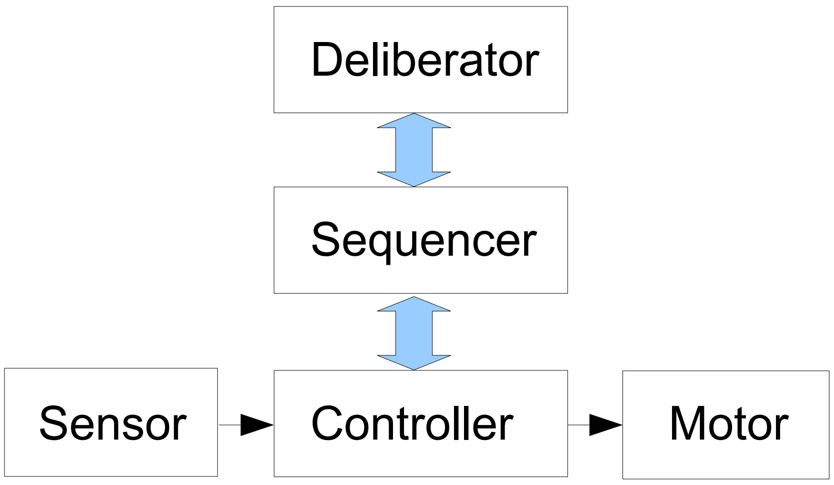


Figure 2.2: Three-Layered Architecture.

create an autonomous robotic agent which involves the concepts of coordination of behavior, coherence of modeling, and communication with other agents. Coordination of behavior means the various basic behaviors must work together in such a way that the goal is accomplished. Coherence of modeling refers to the LPS which must stay up to date to the real world around the robot and most importantly, be an appropriate representation of the environment for the required tasks. Communication is also very important for an autonomous robot since it is rare if ever that such a robot works alone without the need to interact and coordinate actions with another robot or a human.

2.5 Probabilistic Paradigm

Probabilistic models [23] may also be used to control robots instead of behavior based architectures. Designers commonly assume that the physical effects of the control system is deterministic. In actuality, the physical actions of the robot are never ideal and the environment is unpredictable. Such a probabilistic robot incorporates the uncertainties inherent in sensor inputs and physical actions to produce a more robust control system.

Thrun develops a control system using Markov decision processes (MDP), and partially observable Markov decision process (POMDP). Value iteration is used to find the optimal control policy, which uses a payoff function to find the utility of each available control action. MDP control model is developed first since it's simpler to assume the environment is fully observable. In this case, the fully observed state maps to control actions. The control policy maps the best action to the current state that also results from the most likely past states. The policy takes the form of a Bellman equation and all value functions that allow the equation to be solved produce an optimal policy. Replacing an MDP with a POMDP, the fully observable assumption is abandoned for the more realistic partially observable state. The optimal control policy developed using the fully observable assumption only needs a small change to fit a POMDP. The state is simply replaced by a belief, which is a probability distribution over the possible states. The resulting POMDP system is still guaranteed to be optimal.

Probabilistic control systems take into account uncertainty in observation as well as action to produce optimal control actions though the price is high computational requirements. POMDPs are PSPACE-hard problems [21] for finding approximately optimal policies. The only way for POMDPs to act as practical control systems is through approximations and optimizations. For example, the belief space is the large incalculable set of beliefs that is at the center of the computational problem. If the belief space is reduced to only the relevant portions, the remaining beliefs may be computable. This optimization risks the optimality of the control policy to reduce computational requirements.

2.6 Tradeoffs Between Architectures

The SPA and the reactive behavior based architectures described above each have their advantages and drawbacks. SPA is capable of forming a plan to reach its goal but is unresponsive while it is planning. A purely behavioral architecture such as Brook's subsumption architecture respond to changing environments immediately but has no plans or goals other than those found in each individual behavior. Developing a SPA architecture to accomplishing a goal is as simple as giving it the goal and enough time for it to form a plan, but developing a subsumption architecture require trial and error to find the combination of behaviors that accomplish the goal.

Message based architectures are similar to behavior based architectures, consisting of behavior and hardware modules that interact to produce the final behavior. The goal of behavior based architectures is low latency between sensing and acting while message based architectures emphasize extendability. Ideally, message based responds just as fast as behavior based, but it could also be as slow as SPA.

A probabilistic architecture has the advantage of producing the optimal action for each situation. Unlike behavior based architectures, it takes into account the uncertainties of the sensor inputs as well as the actual physical actions. Unfortunately, it also require significant computational resources similar to the planning stage of SPA.

Hybrid architectures take the best qualities of SPA and behavior based architectures to respond quickly and retain the ability to build a plan to reach the goal. The lower levels of the hybrid architecture interface closely with the underlying hardware for responsiveness and act as the reactive behavior based architecture. The planner/deliberator relies on these lower level behaviors to keep the robot out of trouble while the world model updates and the high level plan forms. Naturally, this also require more computational resources than a purely behavioral system.

UBF can take the form of any reactive architecture and be composed of message based and probabilistic components and is a natural fit for the reactive control layer of hybrid architectures. The flexibility of the UBF also promotes reusability of behavior modules and reusability on multiple platforms.

2.7 Biologically-Inspired Robots

In the design of small robots, the inspiration for their form often comes from small creatures, such as insects and worms. The natural habitat of these small creatures are tight, enclosed spaces with uneven surfaces, an environment larger robots find difficult. Imitating a mechanical form known to be natural to that environment shortens development time and creates an intuitive path for improving the design by accurately mimicking the most beneficial part of the form.

2.7.1 $WHEGS^{TM}$.

The $WHEGS^{TM}$ [24] is a series of robots sharing a number of characteristics derived from the cockroach. In particular, the arrangement of legs that give it the tripod gait of a cockroach. To simplify the mechanical design and the control requirements of an articulating leg, the $WHEGS^{TM}$ uses wheel-legs that take the best features of wheels and legs. Wheels are highly mobile on smooth, hard surfaces. However, wheels have difficulty with obstacles on the order of the radius of the wheel or greater. Fully-legged locomotion is better able to traverse difficult terrain but involve complex arrangements of servos and controls. The combination of wheels and legs take the form of several spoke "legs" on each wheel to handle rough terrain without additional servos.

Each $WHEGS^{TM}$ [24] robot includes three pairs of wheel-legs mounted on three axles. Shown in Figure 2.3, each wheel-leg consists of three spoke "legs" set 120 degrees apart. Each wheel-leg is set 60 degrees out of phase from the neighboring wheel-leg and are able to flex out of their original phase to adapt to irregular terrain. One motor drives all three axles, minimizing the weight requirements and control complexity. This design also allow the $WHEGS^{TM}$ to climb over obstacles 1.5 times the leg length by flexing the axle pairs into phase to maximize the torque on the climbing wheel-legs. The arrangement and phase offsets of the three pairs of wheel-legs emulate the motion of the six legs of a cockroach. Following the cockroach's example, the wheel-legs swinging over the body of the $WHEGS^{TM}$ allow it to climb

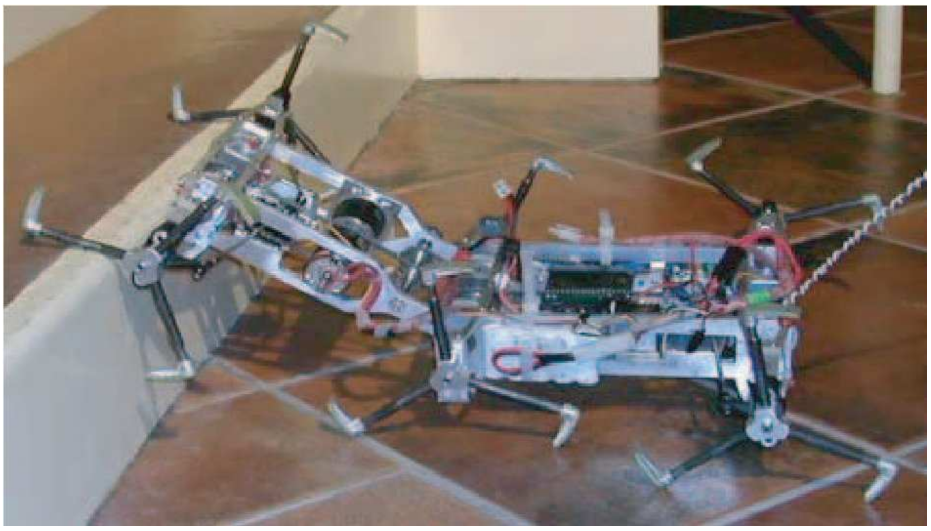


Figure 2.3: $WHEGS^{TM}$ II Rearing Half of Its Body [11].

small obstacles without breaking gait. For turning, the front and rear wheel-legs pivot in opposite directions to minimize the turn radius.

The first of the $WHEGS^{TM}$ [24] series only include the basic features common to the entire series and did not bend its body like a cockroach. From $WHEGS^{TM}$ II on, the series include a body joint at the middle axle. This addition allow the robot to bend the forward torso upward to reach the top of higher obstacles, demonstrated in Figure 2.3, and downward to maintain traction and balance while cresting the obstacle [11]. The $WHEGS^{TM}$ IV, designed to operate near and in the water, is more rugged and is fully enclosed to be water proof and dirt proof. The robot's onboard equipment includes a global positioning system (GPS) receiver, a compass for localization, a sonar for collision avoidance, and a modem communicate with the human operators. The objective of the $WHEGS^{TM}$ IV is for the operator to select a number of waypoints on a map which is then sent to the $WHEGS^{TM}$ through the modem. The onboard control software, running on a microcontroller, drives the robot on the route designated by the sequence of waypoints with a proportional-integral-derivative (PID) controller generating the motor control signals. Different versions of the $WHEGS^{TM}$ robot use different microcontrollers as its onboard processor. The $WHEGS^{TM}$ IV uses a BL2000 microcontroller to execute a PID control algorithm

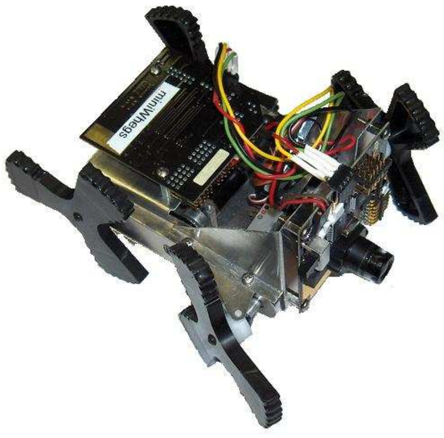


Figure 2.4: A Mini-$WHEGS^{TM}$ Robot.

while the $WHEGS^{TM}$ II uses an Acroname BrainStem [13] for its subsumption architecture controller.

2.7.2 Mini-$WHEGS^{TM}$. Despite being a simple robot mechanically designed to emulate a cockroach, the $WHEGS^{TM}$ is still a relatively large robot at 20 inches long and weighing on the order of 10 to 20 lbs [24]. The much smaller Mini-$WHEGS^{TM}$ robots are designed for reduced size and improved mobility. This series of robots weigh on the order of 100 to 200 grams and are less than 4 inches long [15]. Figure 2.4 shows an example of a Mini-$WHEGS^{TM}$.

The Mini-$WHEGS^{TM}$ [15] is also a series of robots sharing the wheel-leg concept of their larger cousins. The smaller Mini-$WHEGS^{TM}$ included only two pairs of wheel-legs, of which one pair pivots to steer. One motor drives all the wheel-legs like the larger $WHEGS^{TM}$ [24] but only one steering servo is required as opposed to the two steering servos in the $WHEGS^{TM}$. The Mini-$WHEGS^{TM}$ also uses torsional compliant mechanisms, which allow the wheel-legs to twist relative to their axles, and adapt to the terrain they're moving over. A common problem the design encountered is difficulty with certain types of terrain [15]. With versions of wheel-legs that consist of spokes with no footpads, the wheel-legs can penetrate and catch on some surfaces and occasionally fling the robot in to the air. On hard or polished surfaces, the hard wheel-legs have little traction unless the feet are coated in rubber to compensate.

The primary goal of developing the Mini-$WHEGS^{TM}$ [15] from the larger $WHEGS^{TM}$ [24] is to create a low cost robot that can be expendable and can reach places larger robots cannot reach. However, the development of such small robots does not focus on autonomy. Other than remote control by a human operator, most of the Mini-$WHEGS^{TM}$ series only move forward in a straight line until stopped.

2.7.3 RHex. The six legged RHex robot [22], shown in Figure 2.5, is about 0.53 meters long, weighs 7 kg, and is controlled by a PC104 stack with a 100 MHz Intel 486 CPU. The purpose of the design of this robot is to demonstrate a method of locomotion that is comparable to wheels in speed but capable of traversing very rough terrain without complicated mechanisms. The RHex robot has six single jointed legs, three on each side, each powered by a 20 watt motor. Compared with wheels, legs have far more control over the ground reaction forces (GRF) by varying the angle of contact with the ground. However, it is more complicated to control than a set of wheels on a robot. RHex is designed with a control algorithm that tries to maintain three legs on opposite sides of the robot to be in contact with the ground at all times to keep the robot platform stable. When it moves, the rotation of front and back leg on one side of the robot stays in phase with the middle leg of the other side of the robot. To turn while moving forwards or backwards, the rotation speeds of legs on the two sides of the robot is varied, while reversing the rotational direction of the two sides of the robot allows the robot to turn in place. Having an independent motor with its own controller for each leg allows the control algorithm to produce the walking and turning behaviors described above and much more. Also, the sensors on the RHex provide limited monitoring of its body position. Despite this, the control

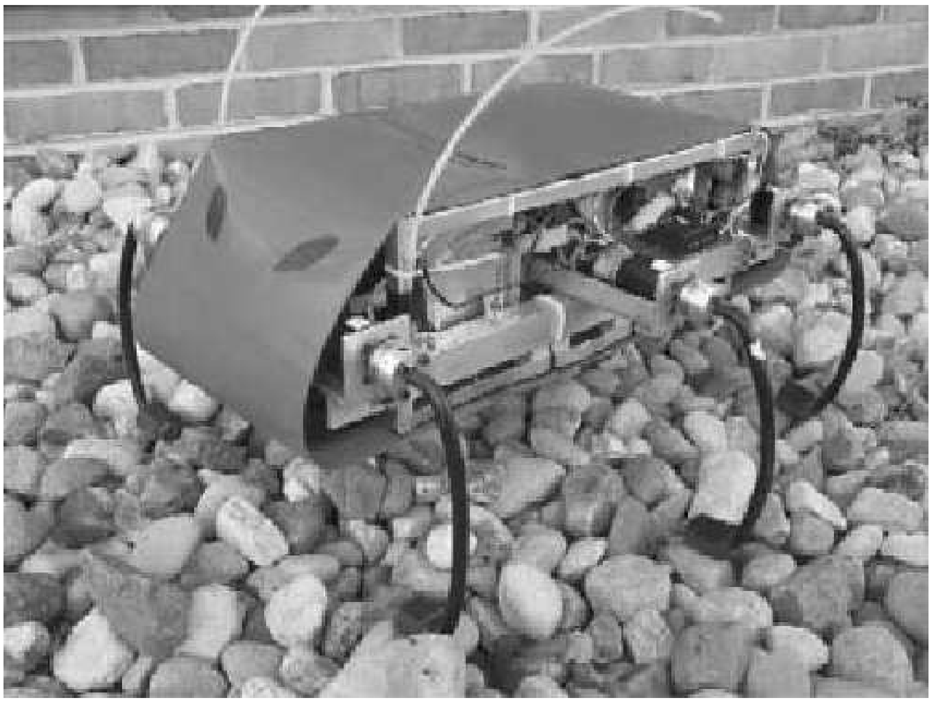


Figure 2.5: RHex Experimental Platform [22].

algorithm and the mechanical structure of the robot is produce a moderately stable physical platform while it moves.

Like the $WHEGS^{TM}$ and the related Mini-$WHEGS^{TM}$ series of robots, the RHex has superior performance over rough terrain. Unlike the $WHEGS^{TM}$, the position of the leg at any one time is essential to keeping the RHex off the ground. The control algorithm creates a "tripod gait" that tries to keep the RHex on its feet. The $WHEGS^{TM}$ on the other hand loses a certain amount of flexibility in behavior with extra spokes per leg but simplifies the control algorithm by not needing to control and maintain specific rotational phase differences between the legs. Where the RHex robot needs to coordinate six motors to walk and turn, the $WHEGS^{TM}$ only needs one control signal for speed and a second for direction.

The RHex has a powerful PC104 stack that does not implement autonomous characteristics for the RHex. This robot appears to only have the most basic behavior modules and a way for human operators to give it movement commands. Its relatively

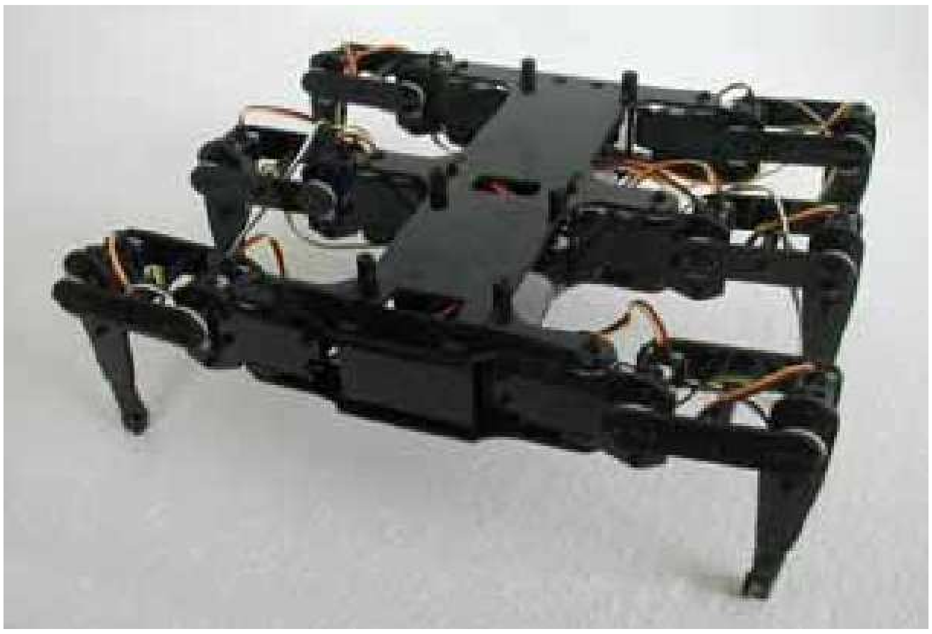


Figure 2.6: Hexapod III Kit Constructed and Wired [18].

complex control system has heavy computational needs despite the fact that it is also relatively simple compared with other legged robots.

2.7.4 Hexapod with IsoPod. Pashenkov [18] explores the use of a new embedded controller on the six-legged Hexapod shown in Figure 2.6. The IsoPod embedded development board convenient processing core for autonomous robots. The board contains a fast, general purpose DSP chip as its processor, a number of I/O options including 12 PWM outputs, and an expansion board that allows up to a total of 22 servos to be controlled by the IsoPod board. The IsoPod comes with a "virtual-parallel processing" operating system called IsoMax that runs the user programmed finite state machines. This makes it very simple to implement and debug simple behaviors since the programming language consists of describing each node and transition of the finite state machines. To complement these features, Brook's subsumption architecture [6] is the control architecture. The subsumption architecture is based on layers of behaviors with higher layers suppressing the output of the lower layers as needed. Each of these layers are modular and can be represented by a finite state machine.

The first version of the subsumption walking controller for this robot is based on a Brooks design [7] for six-legged robots. This design intuitively builds up from the most basic "behaviors" of controlling the position of the leg above the starting ground plane and controlling the position of the leg forward of the starting "relaxed" state. Added to this are higher layers that move the leg up and down, forward and back, a walking module that causes the walking rhythm to ripple through the network of modules, and models that incorporates sensor inputs. This network of modules is perfectly functional but is no longer strictly layered. Another version of the walking controller tested is designed by Porta and Celaya [20] and is capable of walking on rough terrain. This controller is still based on Brooks' design but some modules have been replaced and the layers have been reordered. The resulting controller shows the layering characteristic of subsumption architectures with motor control modules at the bottom and sensing and walk modules at higher layers. The final controller is based on Porta and Celaya's controller. Again, modules were replaced and layers reordered. However, this controller is much clearer with higher level modules distinctly above lower level motor control modules and even makes the two types of motor control distinct (vertical and horizontal movement of the legs).

This IsoPod as a one-chip controller is much more powerful than early attempts with the subsumption architecture and can easily handle the computational requirements of the subsumption architecture. The built in IsoMax operating system also provides a easy way to program basic behaviors using finite state machines(FSM). However, more complex behaviors may be harder or even impossible to describe as FSMs. For example, an IsoPod probably has enough on-board memory to hold a basic model of the environment along with algorithms to control the robot but there's no mechanism to take advantage of it. IsoMax seems most suited to executing the actions from behaviors or low level control behaviors but is also limited by the control architectures the IsoPod can support.

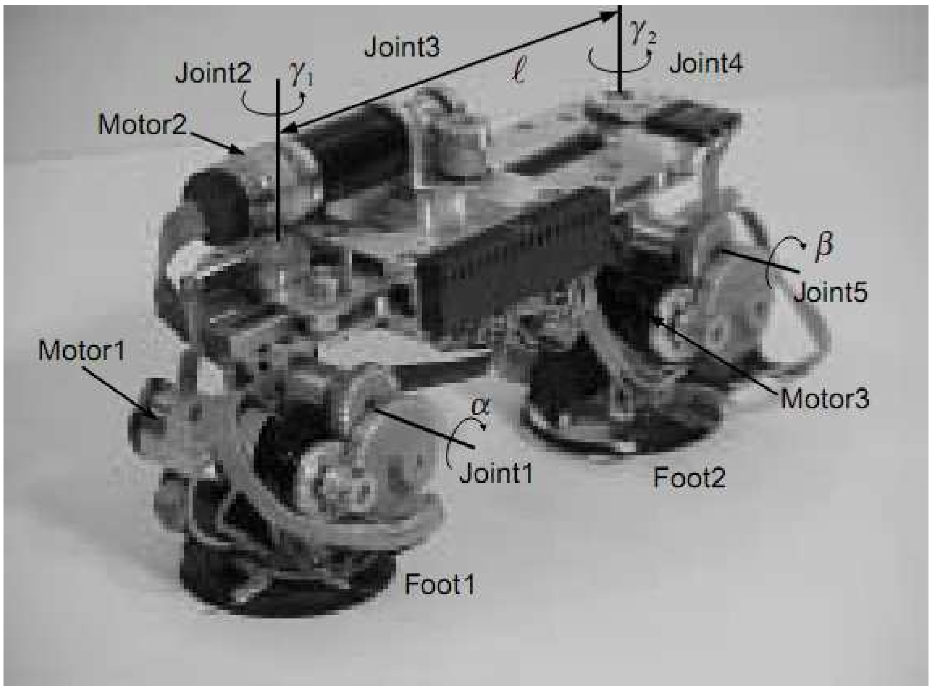


Figure 2.7: Climbing Microrobot [26].

2.7.5 Climbing Microrobot. The climbing microrobot [26], shown in Figure 2.7, is a design that resemble an inch worm. The microrobot is controlled by a Texas Instruments TMS320LF2407 DSP embedded controller and is two legged, about 80 mm long, 50 mm wide, and 450 grams in mass (3.15 in by 1.97 in, 1 lbs.). It is basically two legs with a horizontal component connecting them. The robot is underactuated, which means it has more degrees of freedom (DOF) than number of actuating servos. There are 5 joints in the robot and 3 servos. Joint 1 and 5 bend the pads of leg 1 and 2 respectively and are driven by servo 1 and 3. Joint 2 and 4 rotate legs 1 and 2 and are driven one at a time by servo 2 along with joint 3 which extends and contracts the robot to change the length of the robot and the distance between the legs. Using this hardware configuration, three modes of kinetic operation are implemented: translation, spin 1, and spin 2. The translation mode uses servo 2 to extend and contract the robot. Spin 1 uses servo 2 to extend the robot and at the same time, rotate leg 1. Finally, spin 2 uses servo 2 to contract the robot and at the same time, rotate leg 2. With the 3 modes above and controlling server 1 and 3, three gaits are defined: crawling, pivot, and climbing. Crawling is basically the translation mode of operation and combined with bending joint 1 and joint 5 on leg 1 and 2 where needed to provide full clearance for extending and contracting the robot. This allows the robot to crawl like an inchworm. The pivoting gait is more complicated and uses spin 1 and spin 2 modes and bends joint 1 and joint 2 to provide clearance. The resulting motion is like a stiff-legged crab walk. Since two legs are considered front and back legs, the pivoting gait walks the robot sideways like a crab. The climbing gait is much more complex using the translation mode and joints 1 and 5 to traverse between two intersecting planer surfaces. The robot must start near the destination surface in a contracted state. This is followed by bending joint 1 and extending the body to reach the surface and bending joint 5 match the angle of the target surface. Once the foot pad of leg 2 is secured to the target surface, the rest of the robot can contract and leg 2 repositioned on the target surface by bending joint 1 and 5 again.

The control architecture of this robot centers around the task level scheduler which corresponds to the sequential layer of a three tiered architecture. The task level scheduler takes task level commands given to it and uses a finite state machine to keep track of robot motion status and decompose the command into several motion steps. These motion steps are passed to the behavioral layer where this robot's trajectory planner resolves the inverse kinematic model and interpolates the path to generate a set of desired joint angles. These joint angles are then sent to the joint level controller that sends control signals to the motors and receives feedback from the motors and several other sensors to increase the accuracy of the resulting action. To provide command inputs to the task level scheduler, a human operator could send commands to the command interpreter which outputs task level commands. There is also a motion planner that is like a deliberation layer that takes the initial state, goals, and environmental state to produce commands for the task level scheduler. The motion planner consists of a global planner and a local planner. The global planner finds a possible path that allows an object the size of the robot to fit through. The local planner takes the possible path and produces a feasible path by testing parts of the path that might be problematic to something that moves like this robot such as tight corners of the possible path. If the possible path is found to be not feasible, the local planner requests a new path from the global planner that does not include the problem area. The planner basically does an A* search of the possible paths until a feasible path is found. After a feasible path is found, it is translated into a motion sequence that the task level scheduler can implement.

This microrobot has undergone not only simulation testing but also experimental tests. All gaits function but are limited to smooth surfaces (the kind the vacuum foot pads can attach to). A motion planning simulator tested the motion planning in a software environment before it was tested in a simple maze. The robot is small, relatively simple, and has impressive climbing abilities but is mechanically far more complex than a $WHEGS^{TM}$ [24] robot. The control architecture is very modular. If this robot is modified to have four legs simply for greater payload carrying abilities, only the joint level controller need be modified. Even with additional movement gaits, the task level scheduler may not need updating, just the motion planner to utilize it and behavioral level controllers to direct the servos. The motion planner requires an accurate up-to-date state of the environment to function properly. Given that this is a small robot, it cannot carry many sensors to build an environment model. The accuracy of the map given to the planner is especially important since a few centimeters could mean the difference between a negotiable corner for a feasible path and an area the robot could get stuck in.

2.7.6 Kaa. Another unique robot, Kaa [8] is designed for climbing up and down pairs of pipes. Shown in Figure 2.8, Kaa is a serpentine robot with 13 segments and 12 degrees of freedom. Two 8-bit microprocessors control the robot: one processor control and receive feedback from the servos, the other executes a subsumption architecture [6] controller. The control unit is in the center of the snake rather than one of the two ends to divide the serpentine robot into two tentacles. Servo commands are passed down each segment and acted on in sequence. When the

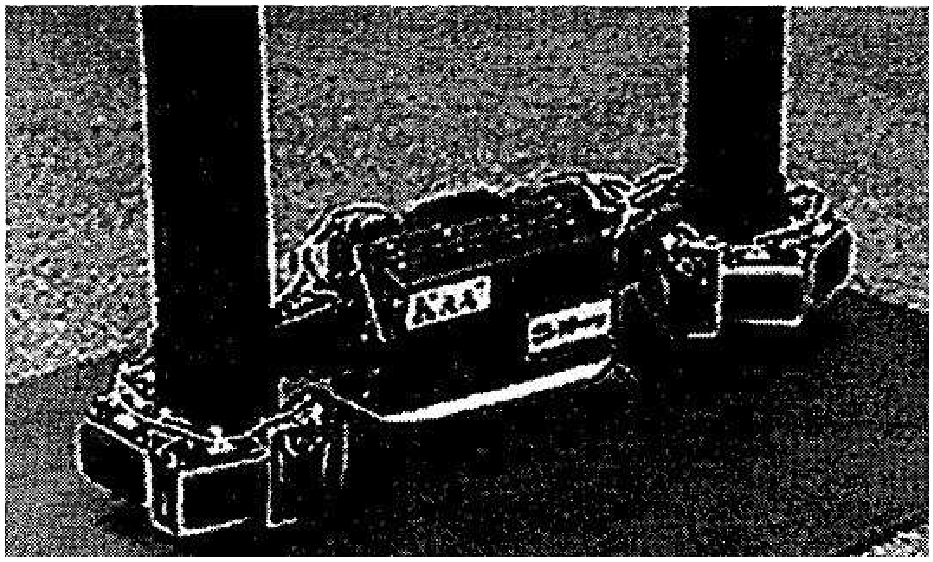


Figure 2.8: Kaa Robot Gripping Two Pipes [8].

central command segment commands a tentacle to grip a pole, the command is sent to the outermost segment to activate its servo to curl the tentacle in the commanded direction. When the movement of the first segment is complete, it passes the command on to the next segment to activate its servos. By the same mechanism, the robot also straightens the tentacles and form traveling waves for locomotion on the ground. This very simple control system also acts like the message based architecture [9] with the potential to add or remove segments for a robot of any desired length.

Kaa is capable of grasping pipes and crawling along the ground using a 32 kB control program. The robot also does not have any sensors to sense the environment other than torque feedback from its servos. The additional sensors would allow it to locate pipes it has not yet made physical contact with and additional degrees of freedom for out of plane motion would allow climbing.

2.8 Overview of the Autonomy of the Biologically-Inspired Robots

Of the six robot systems presented, few had behaviors more advanced than move forward or move to a location. The $WHEGS^{TM}$ [24] rely on human operator input for waypoints. Its only autonomous behavior is collision avoidance enroute to its target locations. Its smaller sibling the Mini-$WHEGS^{TM}$ [15] does not have the GPS receiver needed to navigate to specific locations nor the sensors required to detect obstacles in its path. It simply moves forward until the human operator intervenes. The RHex robot [22] requires a relatively complex control system to position its swinging legs correctly while moving or turning. The system is dedicated to maintaining the walking rhythm but it should be powerful enough to execute a reactive paradigm control architecture on top of the leg control algorithm. The Hexapod [18] already implements the subsumption architecture to control its leg movements. The robot should be able to incorporate abstract behaviors on top of the existing walking behaviors. The climbing microrobot [26] is perhaps the most architecturally advanced of the six. It is similar to Gat's three-layer architecture [10] with a motion planner to build the movement steps and a task level scheduler that make sure they're carried out in sequence. Lastly, Kaa [8] also incorporates a subsumption architecture used to wrap the serpentine robot around pipes or undulate on the ground for locomotion. Again, it's possible for a new higher level behavior to grant this robot more autonomy.

2.9 Summary

Among the different types of control architectures presented, reactive behavior architectures are most like an insect's simple control system. Following a design that mimics insect behavior, responsiveness to the environment takes the highest priority while planning may not even be required. The intended operating environment of the robot is tight quarters with rough irregular terrain. This limits the size of the robot and also limits the choice of control architectures. To minimize size and power requirements on a small robot, a microcontroller is the physical "brain". The limited memory and processing capacity available precludes a probabilistic architecture as well as a three-layered architecture if it is not necessary. Finally, a properly designed message based architecture closely resemble reactive architectures. The advantage of the message based approach would be ease of adding additional components to the robot. Given the size restrictions, the insect like robot does not require such flexibility. The advantage gained from using reactive architectures include building complex behaviors out of hierarchies of simpler behaviors. And of course, the UBF [25] gives flexibility of design and reusability of behaviors in different robots, even large, wheel robots.

Of the robot platforms presented, the climbing microrobot [26] and Kaa [8] are not based on insects. The inch worm like climbing microrobot uses a hybrid paradigm control system to allow it to plan and carry out maneuvers through tight spaces. Its movement mechanism contribute greatly to the need to carefully plan its actions. Although the climbing microrobot is also able to transit from crawling on the ground to crawling up walls, the smooth walls it requires are not in abundance where insects scurry. Speed of movement is more important than the ability to climb vertical surfaces. Kaa resembles a snake and is designed to climb pairs of parallel pipes. Its subsumption architecture [6] allow it to respond quickly and wrap itself around poles when they're detected and ripple across open ground. Unfortunately, neither of these alternatives allow fast movement across the ground like the insect inspired robots.

The remaining robots are either six-legged insectoid robots or are based on the attributes of an insect. While the other three insect inspired platforms do not carry sensors, several models of $WHEGS^{TM}$ [24] include ultrasonic range finders [13]. Their use was inspired by certain crickets and bats which use sound to detect objects. These range finders replaced the need for mechanical antennae or whiskers for detecting obstacles and are able to detect objects at a greater range. Using two ultrasonic sensors set at an 22.5 degrees from directly ahead, 45 degrees apart, an object avoidance behavior autonomously guides the path and speed of the $WHEGS^{TM}$ to avoid collisions. This allows the cockroach inspired $WHEGS^{TM}$ to speed through dark, unlit spaces and maneuver around objects detected by its probing ultrasonic antennae.

III. Design

The goal of this thesis is to adapt the UBF to an embedded processor and to create a Mini-$WHEGS^{TM}$ [15] controlled by the embedded version of the Unified Behavior Framework (UBF). The UBF can execute on the embedded processor, it must interface with the underlying hardware platform. For a practical robot, the hardware platform also includes several sensors in addition to the motors that enable movement. This chapter describes the details of integrating the hardware components with the embedded processor, and with the UBF.

This chapter presents a broad overview of the design, then covers three distinct areas to describe the hardware components of the robot, the software platform that supports the UBF, and the modified UBF, including the demonstration behavior.

3.1 Overview of the Design

The Mini-$WHEGS^{TM}$, a biologically inspired robot, is fitted with sensors and integrated with a modified version of the UBF. This system is low cost, responsive, and capable of performing simple tasks, such as tracking a target, identifying an object, or general exploration. The wheel-legs also allow the robot to traverse rough terrain. This, combined with its small size, allow the Mini-$WHEGS^{TM}$ to perform its tasks in confined areas, like the cockroaches that inspired its design.

The first step to developing the new Mini-$WHEGS^{TM}$ is selecting an embedded processor and connecting the hardware components to it. The Blackfin BF537 microcontroller is selected for its computational resources and having the necessary interfaces to connect the desired hardware components. The availability of a Linux based operating system, uClinux-dist-2008R1.5-RC3, for the Blackfin processor provides a powerful software platform for the UBF to run on the Blackfin. The selected hardware platform also comes with an attached camera and a wireless device server. In addition to this core set of component, there are a pair of motors controlled by bi-directional speed controllers along with rotary encoders attached to each motor, and a set of IR range finders.

The next step is the preparation of the uClinux kernel drivers to access the interfaces connected to the hardware components. The camera requires a complicated driver to initialize and to capture images on request. The motors' speed controllers' accept pulse-width modulation signals which require the *simple_gptimer* driver. To communicate with the IR range finders, the I^2C drivers are needed since the range finders are attached to the I^2C bus. Finally, the *bfin-gpio* driver is extended to enable it to communicate with the rotary encoders.

The remaining step is customizing the UBF to the capabilities of the assembled hardware configuration. To support the integration of the UBF to the platform, a hardware abstraction layer is built to unify the drivers into one interface for the UBF. This provides a separation between the UBF and the hardware platform to make the UBF a multi-platform architecture. On top of this hardware abstraction, a ball seeking behavior is created to demonstrate the UBF's functionality on this biologically inspired robot.

3.2 Hardware Specifications

The embedded computational hardware derives from a Surveyor SRV-1, a small, tracked robot. It is equipped with a microprocessor with several interface ports, a camera, two lasers, a wireless embedded device server, and two rubber treads. Designed for educational and research purposes, the SRV-1 can be remotely operated through a wireless interface or act independently as an autonomous robot. Shown in Figure 3.1, the microprocessor, the camera, and the wireless device server is retained for the Mini-WHEGS while a new set of motors are used. In addition, a pair of encoders attached to the motors keep track of the robot's position, and two IR Range Finders provide information for collision avoidance.

3.2.1 Microprocessor. At the heart of the system is the Blackfin BF537 microcontroller [2]. It is a 32-bit RISC microcontroller operating at 500MHz, with 48 GPIO ports and a variety of interfaces including a I^2C compatible two-wire interface

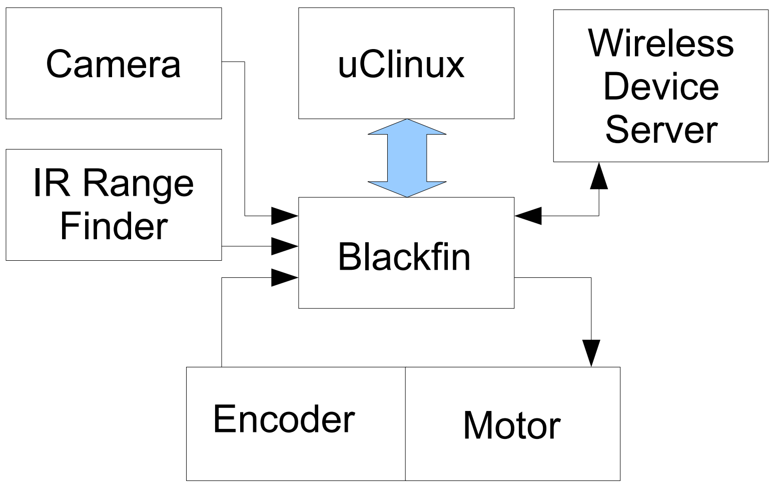
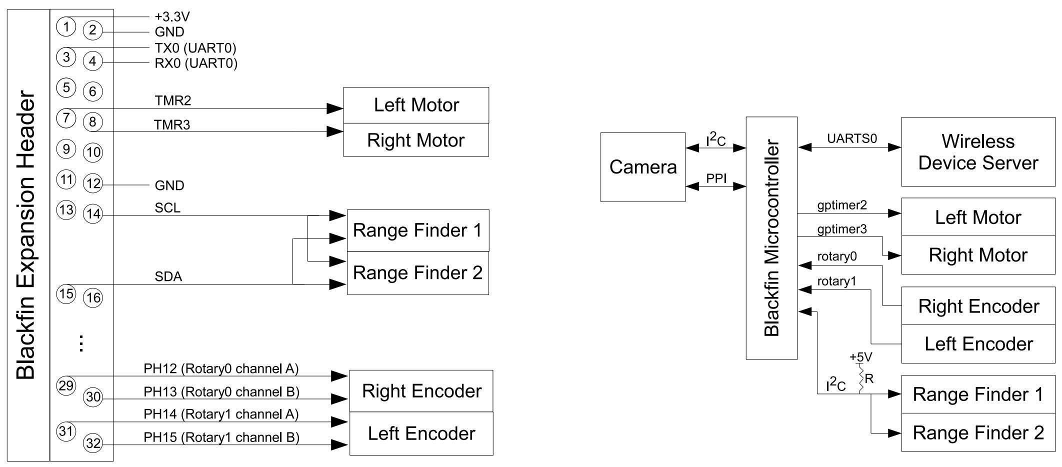


Figure 3.1: System Block Diagram.

(TWI), PPI, and 9 general purpose timers, eight of which produce PWM output. Also attached is a 4MB flash memory device to store instructions for execution on power up, and 32MB SDRAM of memory space for use during execution. Figure 3.2a diagrams how the remaining hardware components are physically connect to the microcontroller, and Figure 3.2b shows how the components are connected from the software point of view. The details of how each component is connected is discussed in their respective section.

3.2.2 Motor. The Mini-$WHEGS^{TM}$ is driven by two motors. Where other variants of Mini-$WHEGS^{TM}$ use a servo for steering and a motor to drive the Mini-$WHEGS^{TM}$ forwards and backwards, this variant uses one motor to drive the pair of wheel-legs on the left side, and the other for the right. Steering is accomplished by skid steering which uses the speed differential between the left and right side wheel-legs to turn while the surface contact points undergo controlled skidding. The motors are each controlled using a bi-directional speed controller, which is controlled with pulse-width modulation (PWM) signals with a period of 20ms and “high” cycle of between 1.5ms and 2.0ms for increasing forward speeds and 1.5ms and 1.0ms for reverse speeds. Shown in Figure 3.2a, the motors’ control wires are connected to two



(a) (b)

Figure 3.2: (a) Component Wiring Diagram. (b) Component Connection Diagram.

of the Blackfin processor's PWM output pins, *TMR2* and *TMR3*. Figure 3.2a shows that *gptimer2* and *gptimer3* are the internal designators for controlling the motors.

3.2.3 Encoder. While the motors drive the robot, an encoder attached to each motor tracks the amount of rotation of the motor drive shaft. Two 1024 count rotary encoders are used, one on each motor. Two digital data channels from each encoder convey the amount and the direction of rotation. The pairs of data channels are set to logical high or low depending on the position of the motor shaft. When the shaft rotates, the signals on the data channels are square waves 90 degrees out of phase with each other as shown in Figure 3.3. A complete rotation produces 1024 cycles of square waves on each data channel. The phase difference between the pair of channels, whether channel A is 90 degrees ahead or behind channel B, tells the direction of rotation. These data channels are connected to two pairs of general purpose IO (GPIO) pins set aside for the rotary encoder. Shown in Figure 3.2a and b, the encoders connect to *rotary0* and *rotary1*, which correspond to the last four pins of Port H. The left encoder is connected to *rotary1* because that results in shorter, untangled wiring connections.

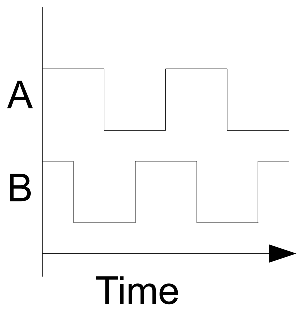


Figure 3.3: A and B Data Channels of the Rotary Encoder.

3.2.4 Camera. The camera is the most important sensor for this basic, insect like robot. It allows the robot to identify and follow objects of interest, flee brightly lit environments, and even receive visual instructions. The OV9655 CMOS camera mounted on the SRV-1 is a color camera capable of capturing YUV and RGB formated images with resolutions up to 1280 by 1024 pixels in 16-bit color. Shown in Figure 3.2b, the processor communicates with the camera through an I^2C [1] interface to control the camera and a special Parallel Peripheral Interface (PPI) [2] to receive the captured image. The SRV-1 positions the connector for the camera so it is securely mounted at the front of of the circuit board, facing forward. Figure 3.2a does not specify the wiring connections of the camera since there is a special connector reserved for it.

3.2.5 IR Range Finder. The IR Range Finder connects to the microcontroller through an I^2C bus as shown in Figure 3.2a, without out the need of pull-up resistors since there are already other devices on the bus. The I^2C connection allow range values to be read from the sensor package's registers. Three sensors are available with three different operating ranges: 4-30cm, 10-80cm, 20-150cm. The range readings are returned in millimeters.

3.2.6 Wireless Device Server. The wireless device server is a wireless connection point that is available for connection from any computer equipped with a wireless

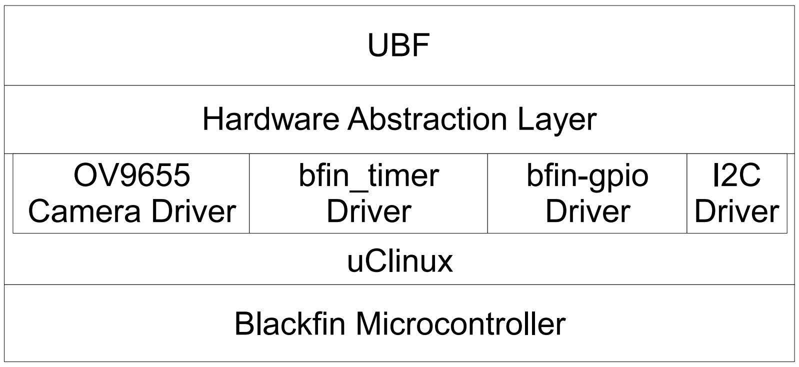


Figure 3.4: Relationship Between the Software Drivers and the Rest of the System.

modem. Shown in Figure 3.2b, the device server connects to the UART0 port of the microcontroller. Its sole purpose is to take messages received wirelessly and pass it through the UART to the processor and send messages received through the UART. Figure 3.2a labels the pin connection points that allow the same communications on a wired connection to the Blackfin.

3.3 Software Platform

A major advantage of the Blackfin microcontroller is the availability of a Linux based operating system. This eases the adaptation of UBF to a microcontroller architecture. Operating in a Linux environment also means the underlying hardware cannot be accessed directly. Fortunately, most of the necessary hardware kernel drivers already exist and only need modifications to fit the requirements for controlling a robot. The collection of disparate drivers are inconvenient to integrate with the UBF so a hardware abstraction layer was created as the single hardware interface for the UBF. Figure 3.4, shows the relationship between the Linux kernel drivers, the user level hardware abstraction layer, UBF, and the Blackfin microcontroller.

3.3.1 Operating System. uClinux [14], a flavor of Linux, is designed for use on microcontrollers in embedded applications. The uClinux-dist-2008R1.5-RC3 used for this robot is designed for Blackfin microcontrollers. After boot up and login, the Linux directory structure is available for access along with many of the standard Linux commands.

Several differences between uClinux and common flavors of Linux exist. The first is the size of the boot image. In embedded applications, the total available memory is always limited. For example, the BF537 system used in this robot has only 4MB of flash memory to boot up from and 32MB of RAM for execution. uClinux is easily configured to add or remove drivers and user applications to control the final boot image size. The required kernel drivers such as *bfin_timer* and *bfin-gpio* must be included to interact with the hardware components. Other drivers such as those for graphics, sound, and USB are not included to reduce the size of the boot image.

Another difference between uClinux and common flavors of Linux stems from the fact that uClinux is designed for microcontrollers that do not possess memory management units (MMU). Linux ordinarily takes advantage of virtual memory to optimize the use of memory during execution and to help maintain distinct memory spaces for each running process. uClinux manages memory usage without hardware assistance and does not use virtual memory techniques. This is normally not an issue for embedded system since they generally do not require multiple processes to execute at once. However, knowledge of this limitation can mean the difference between a successful embedded application and one that runs out of memory. Since memory blocks cannot be remapped as is the case with virtual memory, excessive use of dynamic memory allocation can leave the RAM without sufficiently large blocks of free, contiguous memory to satisfy certain memory allocation requests. The simplest way to avoid this problem is simply avoiding dynamic memory allocation, instead allocate memory statically as much as possible.

The second pitfall is the aggravation of the buffer overrun bug. Buffer overrun occurs when an application attempts to write beyond the end of the allocated buffer in memory, and is a common problem in C/C++ programs where array bounds are not checked. No hardware memory management support also means no memory protection. Common flavors of Linux will stop execution on departure from valid virtual memory space, uClinux does not notice the application is writing to memory outside of its assigned area. If the buffer overrun overwrites some part of memory in use, which is likely given the limited RAM, either data or program instruction is corrupted.

3.3.2 Simple GPTimer Driver. The PWM signals for the motor controls are generated by the *bfin_timer*. This driver allows the setting of the PWM signal's period as well as the width of the high cycle, simply by opening the appropriate device file descriptor and sending an IO command. The standard setting for controlling the motor controller is a period of 20ms with high cycle between 1ms and 2ms where 1.5ms would be zero speed. The lack of documentation led to a full scan of the high cycle width, which determined that a initialization signal of 1.6ms is required as well as the actual zero speed being approximately 1.55ms. Also, a "zero" zone of motor speed is found between 1.62ms and 1.48ms.

3.3.3 GPIO and Encoder Driver. The most basic driver required under uClinux is the *bfin-gpio* driver that is for accessing the GPIO ports. Fortunately, it is supplied in the current of the operating system. With the kernel driver loaded, all subsequent use of GPIO ports are done in user space as low level file operations.

Unfortunately, there is no rotary encoder driver for BF537. An interrupt handler is added to the *bfin-gpio* driver to monitor two pairs of GPIO, *gpio44* and *gpio45*, and *gpio46* and *gpio47*, for the encoder functionality. The interrupt is set to trigger on both the rising and the falling edge of each data channel and maintain individual counts. When 4 events are counted for both data channels in a pair, the encoder records one tick (with 1024 ticks per motor rotation). Initially, this driver was designed to identify reversal of direction of rotation based on the interrupt event counts and automatically track direction of movement as well as maintain the distance count centered at the start location. This proved to be unreliable as there are false trigger events that randomly reverses the direction of movement. To maximize accuracy, the user supplies the expected direction of movement to the encoder driver to enable the driver to ignore false trigger events. When the user application, the UBF in this case, commands the motor to rotate, the direction of rotation is also given to the rotary encoder driver. The encoder driver then resets the tick counts to measure the linear distance traveled in the given direction since the last direction change. This simpler rotary encoder driver provides accurate rotational tick counts in the direction of the last user command and leaves the rest of odometry based position tracking to be handled at the user level.

3.3.4 Camera Driver. The OV9655 camera driver was found to be in a partially functional state. It contained enough instructions to prepare the I^2C and PPI connections to the camera and initialize it to capture 1280 by 1024 pixel images. However, it did not respond to any other commands since they had not been programmed. Using the OV9655 camera datasheet as well as the SRV-1's default firmware source code as a reference, a complete set of camera control codes are added to set the camera similar to the functional firmware camera control code. The camera has numerous settings that can be set through the I^2C interface. The complete set of camera control codes found in SRV-1's firmware source code initializes the camera and prepares it to capture images in YUV format. Using the OV9655 datasheet, additional control code sets are added for changing the camera's image capture resolution and capture format. In actual usage, the camera is set to return 16-bit RGB formatted images at 1280 by 1024. Any further processing of the captured image, such as down sampling, is done at the user level.

3.3.5 I^2C driver. The I^2C driver is a selectable kernel driver in uClinux. Although the primary use of the driver is to communicate with the IR Range Finders, it can also communicate with the camera. To access the I^2C bus through this driver, the user level application uses low level file operations to read from or write to specific registers of the device specified by an I^2C device address.

3.3.6 Wireless Communications. Wireless communications is extremely simple if the messages contain only ASCII text. The wireless device server allows a socket to connect to the device and communicate in messages in plain text with the microprocessor. Although the system cannot initiate wireless communications, once an external device connects to the wireless device server, communications is as simple as reading from and writing to the console without requiring a software driver. For example, in C, the *printf* and *fgets* functions are all that is needed communicate through the wireless device server.

3.4 UBF on Blackfin

Porting UBF to the Blackfin BF537 running uClinux requires writing a new user level hardware driver to facilitate interfacing the "generic" UBF to the particular platform and writing behaviors appropriate to the capabilities of the robot and the desired task goals. Memory management and memory size can be problematic, which requires good programming practice to prevent their occurrence.

3.4.1 User Level Hardware Driver. The user level driver organizes all the necessary calls to initialize and operate devices and collect them together to create one simple interface to the underlying hardware. Two new classes are created. The first act as a Hardware Abstraction Layer (HAL) to provide standard functionality to the UBF without needing to know the nature of the hardware platform. The second is a camera class within the HAL class designed to initialize the camera as well as hold the helper functions for processing the captured image. Figure 3.5 gives a visual description of the relationship between the HAL class named *miniWHEGS*, the rest of UBF, and the hardware it controls.

The *miniWHEGS* class prepares the underlying hardware for use by the UBF and groups all access functions together to form one monolithic interface. At startup, it initializes the two PWM generators for controlling the motors and the two rotary encoders to track the rotation of each motor. The class also exposes access functions

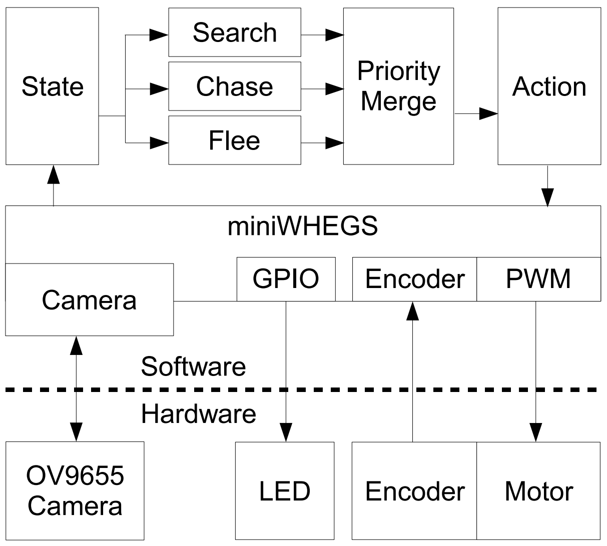


Figure 3.5: Relationship Between the UBF and the Hardware Components.

for controlling the LEDs and for retrieving the change in position and the direction the robot is facing. This is also where the physical dimensions of the robot are set which are required for computing the physical position and orientation from the encoder tick counts. The motor controls are abstracted as a turn command and a speed command which are converted within the HAL to speed commands for the left and right motors. Skid steering allows maneuvers such as turning in place that pivoting wheel-legs cannot achieve. The speed and turn commands are both abstracted to the range of values between +100 and -100. Positive speed is forward while negative speed is reverse, and positive and negative turn values are left and right turn respectively. Abstracting the steering capability to turning and forward or reverse speeds allow greater compatibility between the UBF for the skid steering Mini-$WHEGS^{TM}$ and other robots, including other versions of Mini-$WHEGS^{TM}$.

The *camera* class is a separate object inside the *miniWHEGS* class to contain the complexity of operating the vision system. The class holds all the variables needed to operate the camera and its image processing functions. The initialization function holds the full set of calls necessary to prepare the camera for use and start capturing

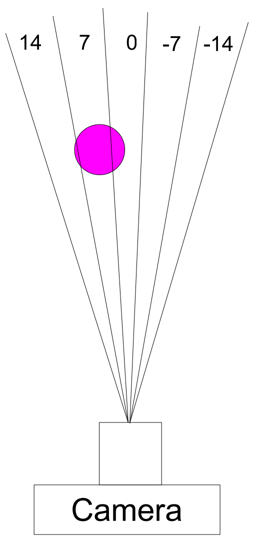


Figure 3.6: Visual Representation of the *seekcolor* Algorithm.

images. The primary use of the camera is to find the relative position of objects of the desired color range in the horizontal plane. To support that use, the camera class includes subsampling to reduce computation time and a simple 1-dimensional color concentration detector. The subsampling function accesses the image plane at a lower resolution index count and skipping pixels. By default, the subsampling algorithm skips 8 pixels in both the x and y axis, picking out a sparse matrix of 160x128 pixels from 1280x1024 pixels. After using the subsample function to obtain a reduced image plane, the color concentration detection function called *seekcolor* scans the image for pixels of the desired color range defined by upper and lower RGB bounding values. Depending on which of 5 vertical bins of pixels the desired color is found, the count is tallied to find the approximate angle to the object. The center bin is 0 degrees deviation to the object, the inner bins indicate a deviation of 7 degrees, and 14 degrees for the outer bins. This is represented visually in Figure 3.6. The 5 triangular fans of the divided field of view each contain a certain number of pink pixels, which in this case is most concentrated at 7 degrees to the left of center. The pixel count

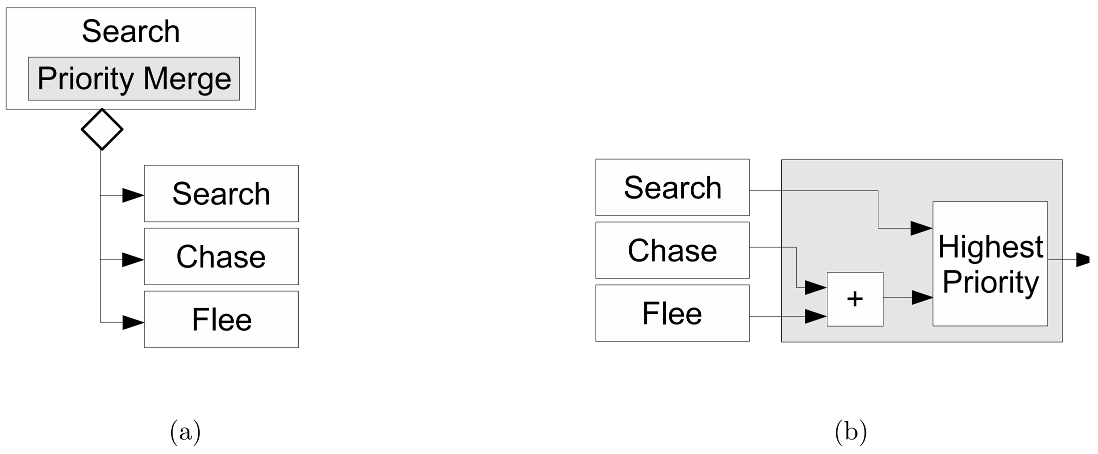


Figure 3.7: (a) Structural Diagram of the Ball Seeking Behavior (b) Functional Diagram of the Ball Seeking Behavior.

also gives a sense of distance to the target. Since the target has a fixed size, it has predictable pixel counts at varying distances away from the camera.

3.4.2 Seek the Big Pink Ball. The goal of this robot is to simulate insect behavior. This set of behaviors chase a bright pink object but backs away if the object is too close. If no bright pink object is in view, it searches for the object. Shown in Figure 3.7a and b, three separate behaviors combine to produce the final behavior: *search*, *chase*, and *flee*. Figure 3.7a shows the structure of the composite behavior and Figure 3.7b gives a sense of how the *prioritymerge* treats the three behaviors.

The first behavior module is *search* and produces an action only when no pink pixels are found. When active, the behavior initially maintains the previous turn command. A counter included in the *state* object allow the behavior to periodically choose a random turn direction, at which time, the Mini-$WHEGS^{TM}$ begins turning in a small circle. The *search* behavior produces a movement command with forward speed of +10 and turn value of ±80. Upon detecting pink pixels, *search* deactivates and allow *chase* to turn and move the Mini-$WHEGS^{TM}$ toward the pink ball. The *chase* behavior tries to keep the ball in the center of the visual field while moving the robot closer to the target by setting a constant forward speed of +40 and turn value of 0, ±30, or ±60, depending on if the detected pink pixels are concentrated in the center bin, inner bins, or outer bins. As the pink ball grows larger in the visual field and the pink pixel count climbs, *flee* inhibits the forward driving command of *chase* without a turn command and eventually bring the robot to a halt at a comfortable distance from the pink ball. Starting at a pink pixel count of 100, the *flee* behavior produces 0 speed. With increasing pink pixel count corresponding to a closer pink ball, the behavior produce increasingly negative speed up to -80. The comfortable distance where the combination of *chase* and *flee* speed commands stop the robot occurs at the distance where the highest pink pixel count out of the center bin is 140 pink pixels. If the pink ball grows even larger in the visual field, that is, it moves to within the comfort range of the Mini-$WHEGS^{TM}$, *flee* overcomes the forward drive command of *chase* to back the robot away from the pink ball at a maximum of -40 speed to maintain a safe distance from it. All three behaviors seek the same pink ball between the RGB color values of [0, 20, 40] and [255, 140, 180].

The *search* module operates by superseding other behavior modules when a specific condition is met, which is when no pink objects are detected. The *chase* and *flee* modules complement each other. The *chase* module keeps the pink ball centered in the visual field and the *flee* inhibits and overcomes the forward movement of *chase* when necessary. The arbiter appropriate for these three behavior modules is the *prioritymerge* arbiter.

3.4.3 PriorityMerge. The characteristics of *prioritymerge* are that the highest priority action is selected and equal priority actions are summed. This is very similar to *highestactivation* where the highest value action is selected. However, *highestactivation* is not designed to handle multiple equal valued actions and will only select the first of several highest valued actions. The *prioritymerge* arbiter handles the situation by summing the equal valued actions of the highest value found. Figure 3.7b indicates that *chase* and *flee* have equal priority by design, and competes with *search* for highest priority together.

3.4.4 Customized State and Action. The last of the Mini-$WHEGS^{TM}$ specific changes to UBF are a new derived *state* object called *state_miniWHEGS* and a derived *action* object called *action_miniWHEGS*. Both of these objects contain additional functions and variables to allow access to motors, encoders, camera, range finder, and wireless communications defined through *miniWHEGS*. Functions in *state_miniWHEGS* provide processed odometry data in x, y, and facing angle. It also provides direction and pixel count for the desired color, range to obstacles detected by the IR Range Finders, the current set speed and turn rate, and last message received through the wireless device server. Functions in *action_miniWHEGS* allow behaviors to set the seek color range, the speed and turn rate, and set messages to be sent though the wireless device server.

3.5 Summary

Implementing UBF on a new physical platform involve designing the hardware configuration, a hardware/software interface, and UBF state and action objects extended to take advantage of the hardware. The hardware configuration must be capable of supporting the task goals. This includes at a minimum motors to move the robot around in the physical world and sensors to take input from the environment. On top of that, a Linux based operating system provide a consistent target development platform for programmers, as well as drivers for the hardware components. Integrated at the lowest level of the modified UBF is a hardware abstraction layer to provide a consistent hardware interface for the generic UBF. Finally, the generic state and action objects are extended to match the capabilities presented by the HAL. All of these changes allow behavior modules to be compatible between UBF running on the Mini-$WHEGS^{TM}$ and UBF running on similarly configured hardware platforms.

IV. Results

As a biologically inspired robot, the Mini-$WHEGS^{TM}$ [15] with the Unified Behavior Framework (UBF) is drawn to and fears the pink ball much like a moth is drawn to a flame yet fearing the heat when it gets too close. Achieving this behavior requires that each hardware component be characterized so they can be properly integrated to the software. As one coherent system under the HAL and UBF, the Mini-$WHEGS^{TM}$ exhibits lifelike behavior which demonstrates the effectiveness of the UBF on the embedded system, and meeting the biologically inspired goal.

This chapter characterizes each hardware component as they are integrated into the hardware abstraction layer (HAL). This is followed by a presentation of the observed behavior of the UBF customized for the Mini-$WHEGS^{TM}$ [15] in several test scenarios. These scenarios test the response of the robot in a environment with the ball in a fixed position, and in a dynamic environment where it responds to a moving ball.

4.1 Hardware Development Results

Four components underwent significant development to integrate into the HAL. The PWM motor control was simple in theory but still required a brute force solution to determine the proper control protocol. The rotary encoder required an expansion to the existing GPIO driver. The camera driver, while it was preexisting, it was not complete. Lastly, the IR range finder is not be detected on the microcontroller's I^2C bus and remains unintegrated.

4.1.1 PWM motor control. Each motor is controlled by a speed controller, which responds to a PWM control signal. Ideally, the same PWM signal sent to both speed controllers results in both motors rotating at the same rate. In real life, slight differences results in one motor rotating slightly faster than the other. In this robot, the right motor is slower than the left motor and has a larger "zero speed" zone around the actual zero speed. Since the HAL remaps UBF motor commands to PWM signals for the speed controllers, careful characterization of the command response of each motor allows the HAL to compensate for the speed difference between the two motors.

4.1.2 Rotary Encoder. The rotary encoder requires additional functionality to be implemented in the *bfin-gpio* driver. The rotary encoder output signal is well documented and is received by setting specific GPIO ports to count the rising and falling edges of the rotary encoder signals. During hardware tests with the rotary encoder driver, false signal edges made automatic detection of the direction of rotation impossible. Ideally, the encoder keeps track of the amount of rotation as well as the change in direction of rotation as an independent feedback of the movement commands. Since the direction of rotation cannot be reliably tracked through the encoder, the HAL stores the movement direction of the last action command and use it to calculate the position and pose of the robot. If an external force push the robot in such a way that the wheel-legs rotate opposite the expected direction based on the movement command, the real movement of the robot would not be accounted for correctly.

4.1.3 Camera. Preparing the camera for use with UBF involve finding the color range of the bright pink ball the Mini-$WHEGS^{TM}$ is to seek. By capturing an image from the camera and examining the RGB color components of the the pink ball in the visual field, upper and lower bounds for the acceptable color range is found. However, the exact relations between RGB values that differentiate between colors is more complicated. The algorithm used to determine if a pixel is pink first checks if the RGB components are between the low limit of [90, 20, 40] and the high limit of [255, 120, 160] for the red, green, and blue components respectively. This does not adequately differentiate pink for similar yet visually different colors such as orange. Adding a second step to check that the red value greater than the blue component, and that the blue component is greater than the green component yields more reliable results. However, the determination of color is still highly dependent on the lighting

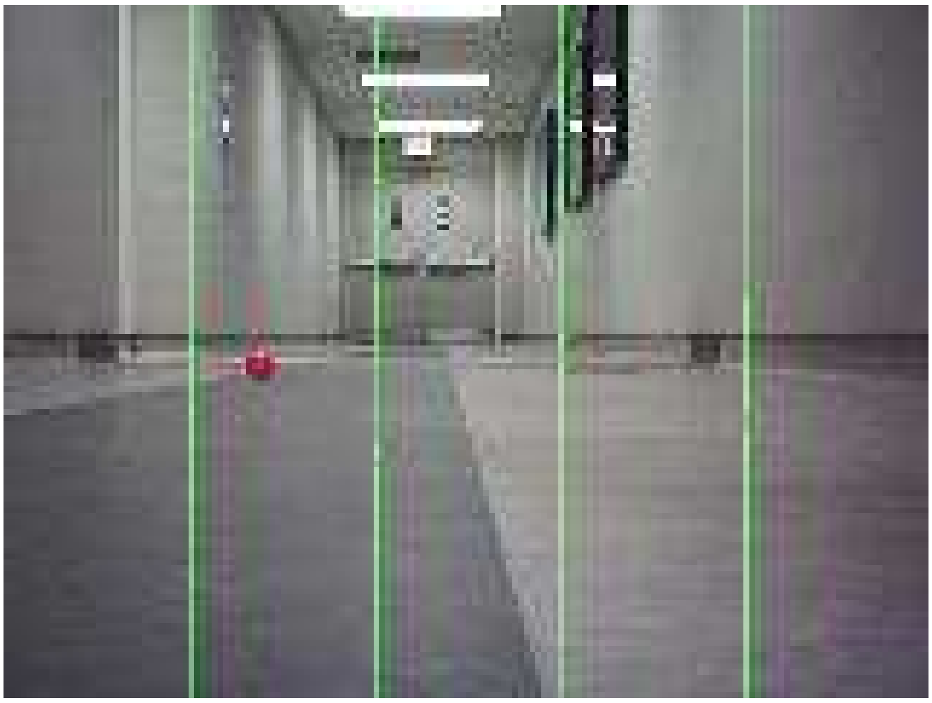


Figure 4.1: Mini-$WHEGS^{TM}$ Sees the Pink Ball at 360cm

conditions. Perhaps a different color space can better differentiate between colors that are distinct to the human eye.

Other characteristics of the camera that greatly affect the behavior of the robot are field of view, visual range, and response time. The field of view of the camera is found to be only about 35 degrees wide. The camera can detect objects directly in front of it but casually waving the object in front of the camera easily moves it in and out of the field of view. The UBF designed for this robot compensates for the intermittent loss of tracking by storing and continuing the last turn command with the expectation that the pink ball is just outside the narrow field of view in that direction of the turn. The visual range of the robot is greater than is needed for operating in small enclosed areas. For an object the size of the pink ball, which is about 6.5cm in diameter, the camera can distinguish the ball at 360cm away. This is shown in Figure 4.1, which is an image of the ball at approximately 360cm away, captured by the camera, and down sampled to 160x128 pixels. However, the detection of pink pixels start to become sporadic at this range from the larger proportion of white glare, and the dark, unlit underside of the ball. The minimum visual range of the camera partly results from the camera's physical location on the robot and partly from the narrow field of view. Shown in Figure 4.3, the ball starts to disappear below the camera's field of view at just over 30cm from the front of the camera. With a proper color discrimination algorithm, the pink ball is distinguishable from the extreme range of 360cm to 0cm where the ball is in physical contact with the robot long as the overhead lighting does not leave such a large glare on the top of the reflective ball that it appears white. Also shown in the captured images such as Figure 4.1 are the 5 bins of the *seekcolor* algorithm divided by vertical, green lines. Finally, the time-delay associated with capturing an image from the camera directly affect the action of the robot. The time delay of the image capture process is approximately 0.11s. This occurs each time the state object updates so actions are generated based on 0.11s old image data. Other sensors, such as the IR range finder, would provide the necessary information about the environment in between camera image captures, and strategic use of multi-threading would prevent generated actions from being delayed as well.

4.1.4 IR Range Finder. The IR range finder promised to be simple to integrate into the system since it uses the I^2C protocol. The Blackfin processor supports I^2C communications with dedicated pins for it. The uClinux operating system also supported it with an I^2C kernel driver. After finding the correct method to physically connect the device to the I^2C bus, the device is detected on the bus, and distance measurements are read through the use of the I^2C device driver. It must be noted that since this is a bus, each individual device must be set with a different device address so they are distinguishable on the bus. Although the readings are easy to obtain, they also appear to be deviate 5 to 10 percent from measured distance. The surface property of the target object may be the source of this error along with the angle of incidence with the surface.

4.1.5 Summary of Hardware Development Results. Of the three hardware components that were successfully integrated, all three require the HAL to compensate for deficiencies. The PWM motor controls are capable of controlling the motors but require additional work to synchronize the rotation speed of the motors. This is a situation where a reactive control architecture can respond quickly to the environment and course correct when the robot is not heading straight toward its target. However, being able to move in a straight line without a clear target to aim for is a better physical platform. The rotary encoder and the camera both have performance feature deficiencies. The rotary encoder cannot automatically identify the direction of rotation of the encoder. Instead the HAL records that information as well as calculate the odometry from the encoder outputs. The camera is useful for identifying color targets. Other than capturing the image, all other processing is done in the HAL.

4.2 UBF in Action

The completed Mini-$WHEGS^{TM}$ is a shy and fearful creature. Four test scenarios reveal the real-life behavior of this creature and also its physical limitations. The first and second scenarios are related with the ball left in a fixed location for the robot to find and stare at. The third scenario involve moving the ball out of sight whenever the robot sees it. The final scenario keeps the ball within the robot's field of vision. Since there is no range finder mounted on the robot, the tests are conducted in an open, uncluttered area to avoid unnecessary collisions.

4.2.1 Starting with the Ball in Sight. The first scenario starts the Mini-$WHEGS^{TM}$'s autonomous behavior with the ball far away but within its field of view, similar to Figure 4.2. The *chase* behavior is expected to immediately turn and move the robot toward the pink ball. Soon, the *flee* behavior would slow then stop the robot at a safe distance away from the pink ball.

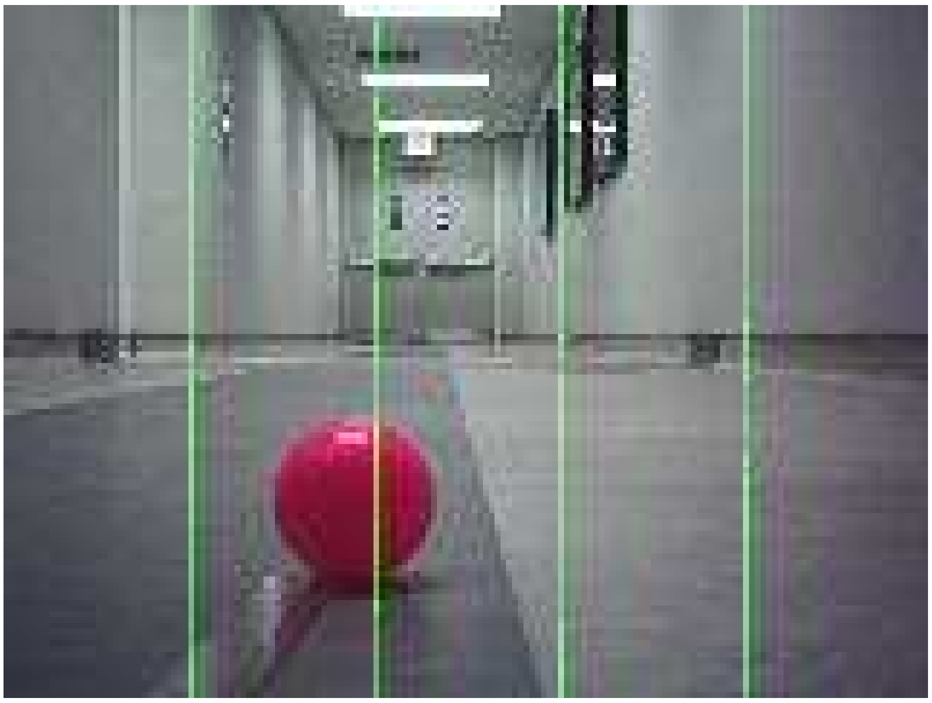


Figure 4.2: Mini-$WHEGS^{TM}$ Sees the Pink Ball at 60cm

4.2.1.1 Observed Behavior. Upon activation, the robot quickly turns to point at the pink ball as it accelerates. Closing on the ball, it slows down then stops to stare at the ball.

4.2.2 Starting with no Ball in Sight. In the second scenario, the autonomous behavior is activated with no ball in sight. The pink ball is placed several feet away behind the robot. The *search* behavior module is expected to dominate immediately and turn the robot in small tight circles to look for pink pixels that indicate the pink ball. Upon sighting the ball, *chase* and *flee* would bring the robot to a safe distance from the ball before stopping.

4.2.2.1 Observed Behavior. When the Mini-$WHEGS^{TM}$ awakes and sees no pink pixels, it sits still for a time since there was no prior maneuver. When the counter elapses, it randomly chooses to either turn left or right in a slow, tight circle to reduce the amount of skidding that would result from an in-place skid turn. The robot soon turns far enough to see the pink ball at the edge of it's field of vision and increases forward speed as it continues to turn toward the ball. Finally, it comes to a stop facing the ball.

4.2.3 Keep Away. The third scenario again starts with the pink ball out of sight. This time, the ball starts to the side of the robot. The *search* behavior would turn in tight circles to look for it. If it's lucky, it finds the ball quickly and turns to move toward it. This is when the ball is moved across and out of the robot's field of vision to the side. The expected behavior is that the *chase* behavior tracks the relative angle to the ball as the ball is moved. When the ball is out of sight again, *search* continues the last turn command for a time before turning in a random direction.

4.2.3.1 Observed Behavior. Starting with no pink ball in sight, it sits still for a time before turning in a tight circle. The ball is soon discovered sitting just to the side of the starting field of view. Immediately, the ball is moved at ground level where it is guaranteed to be visible to the robot while it turns to try to keep the ball in sight. The ball is moved far to the other side of the robot, out of it field of view. The robot continues to turn toward the last know direction to the ball and soon sights and homes in on the ball. The test continues with another rapid displacement of the pink ball. This time the ball is kept moving, out of sight of the robot for a longer period of time. The robot conducts a *search* along the last known direction to the ball until the counter elapses and the robot chooses a new random turn direction. This time, it turns away to look for the ball in the other direction.

4.2.4 Dance with the Ball. The final test scenario starts with the pink ball front of the robot where it's too close for comfort. Figure 4.3 shows what the Mini-$WHEGS^{TM}$ sees at the start of this scenario. The robot is expected to start turning toward the ball and backing away until the pink ball is at a safe, comfortable distance away. The ball would then be moved slowly enough for the robot to turn and track. The ball is also moved closer to and farther from the robot, which should cause the

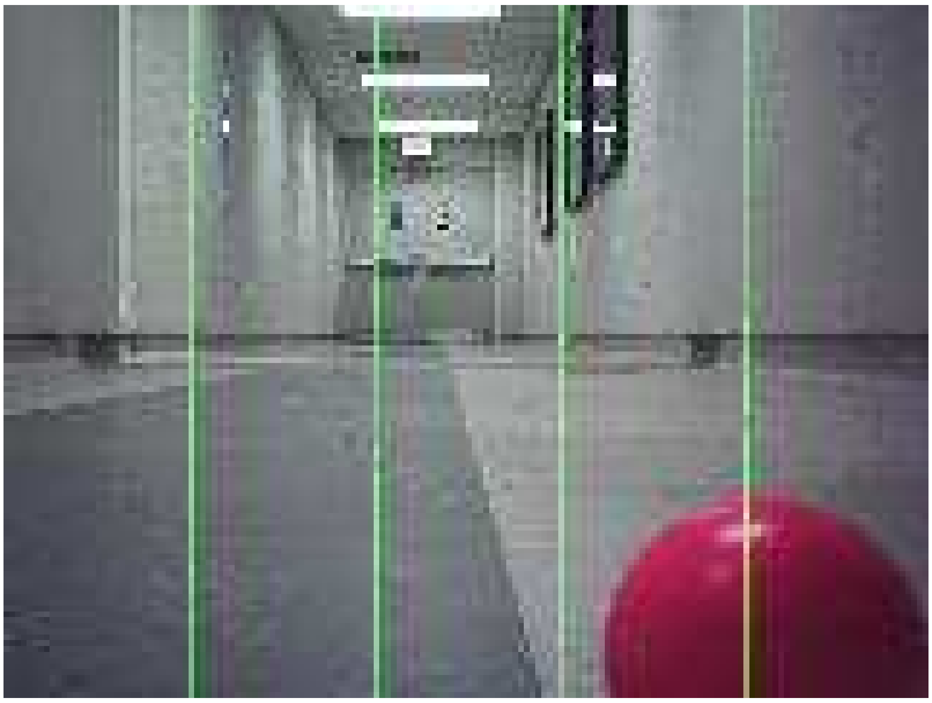


Figure 4.3: Mini-$WHEGS^{TM}$ Sees the Pink Ball at 30cm

robot to approach and back away as it tries to maintain the comfortable distance between the robot and the ball.

4.2.4.1 Observed Behavior. The Mini-$WHEGS^{TM}$ immediately backs away as it turns to center the ball in its view. Moving the ball forward to keep it close to the front of the robot forces it to continue to back away at speed. When the ball is suddenly moved farther away but kept in the robot's field of view, the robot reverses course and leap forward to stay with the ball while turning to keep the ball centered. Waving the ball around in front of it quickly only cause the robot to move forward and back if the waving is not drastic enough. Large movements of the ball cause the robot to turn to track it as it continues to move forward and back since the pink ball also appear to grow and shrink in its eye.

4.2.5 Summary of Behavior Tests. Four test scenarios demonstrated the functionality of the UBF as well as the microcontroller and the Mini-$WHEGS^{TM}$ robot. The first two scenarios showed the behaviors are stable in situations where the environment isn't constantly changing. The remaining two clearly show its ability to operate in a rapidly changing environment.

4.3 Summary

Built from a set of sensors, motors, and a low lying frame, the Mini-$WHEGS^{TM}$ takes the form of a crawling critter. After integration under a hardware abstraction layer, and controlled by the Unified Behavior Framework, the Mini-$WHEGS^{TM}$ also act like a living creature, one who's life revolves around a pink ball. It searches all around for the pink ball when it does not see it, it is drawn to the ball when it does see it, and yet, it is afraid to get too close to it. By moving the ball around within its field of view, the robot can be guided to any position in its environment without using direct means to give it commands.

V. Conclusions

This thesis demonstrates the feasibility of a flexible, autonomous control architecture on an embedded system. The Mini-$WHEGS^{TM}$, designed to mechanically mimic the cockroach, now behaves like a shy, fearful cockroach. This chapter summarizes the results of the research, discusses possible future work, and ends with additional remarks about the development of the Unified Behavior Framework (UBF).

5.1 Research Conclusions

The objectives of this research is to adapt the Unified Behavior Framework to an embedded platform, and make the Mini-$WHEGS^{TM}$ fully autonomous in a task of locating and following a target. These goals are accomplished using a flavor of Linux designed for microcontrollers, a microcontroller with enough IO resources to interact with the Mini-$WHEGS^{TM}$ hardware components, and linking the UBF to the hardware abstraction layer (HAL) designed for the Mini-$WHEGS^{TM}$.

The availability of Linux on microcontrollers greatly simplified the development process of UBF for an embedded platform. Reviewing the existing UBF designed on a desktop and modifying it with respect to the limited resources of an embedded platform is sufficient to create UBF for embedded platforms. The key differences to watch for is the lack of a hardware memory manager, and the general shortage of operating memory. The former calls for limiting if not eliminating the use of dynamic memory allocation, and the latter demands efficient use of memory.

The integration of hardware components with Linux proved to be more complex. While Linux eased the transition from a desktop computer, the operating system also require drivers before a user process like UBF can access the hardware components. Once the hardware components can be accesses, the full set of sensors and motors must be integrated into a hardware abstraction layer. The HAL keeps the details of interacting with the hardware platform out of the rest of the UBF. It also prevents the UBF from being locked to a specific hardware platform.

All details particular to the hardware platform are contained within the HAL. Specifics such as the dimensions of the robot is required to compute the motion model. Another important detail is the motor control signals required to drive the motors. Not all motors and speed controllers are equal but they're all equal through the HAL.

5.2 Future Work

The UBF showed the importance of standardized interfaces between behavior modules. While the behavior modules receive sensor input and effect physical actions through the *state* and *action* objects, there is no standardized interface between these two objects and the underlying hardware. The HAL bridges this gap in an attempt to keep this version of the UBF from being tied to this Mini-$WHEGS^{TM}$. Further development of the HAL, perhaps integrating capabilities and features from similar solutions, may allow the embedded UBF to be fully compatible with the "full size" version of the UBF.

Another area for continued development is the I^2C bus. It has the most potential for integrating sensors that only require moderate bandwidth. The bus allows many devices to be connected regardless of the number of available IO ports and physical pins. Although there is a restriction on power usage and space on and around the robot for mounting sensors, managing a single data bus is simpler than managing a array of different IO ports, each with their own protocol.

The OV9655 has a field of view of approximately 35 degrees. A different camera or perhaps the addition of a lens so that the field of view is much shorter and wider may produce better results. This robot is designed more for enclosed spaces where peripheral vision is more useful than long range sight. A second camera may also give interesting enhancement to the robots capabilities. With two cameras set to maximize the combined field of view, a rodent becomes an appropriate model for behavioral development.

5.3 Final Remarks

Low cost, embedded, autonomous systems were once words that did not belong near each other. Either low cost systems are not powerful enough or the autonomous system is too large to be considered "embedded". With processing power increasing by leaps and bounds, low cost embedded platforms are practical for autonomous systems. Such powerful embedded systems are enough for many simple desktop applications. With an eye for optimization, even something as small as a Mini-$WHEGS^{TM}$ can do everything a much larger robot can do. Certainly, UBF behavior modules designed for larger robots can work on the Mini-$WHEGS^{TM}$ with little or no change.

Bibliography

1. "The I^2C-bus specification version 2.1", January 2000.
2. "$ADSP-BF537BlackfinProcessorHardwareReference$", December 2005.
3. Arkin, Ronold C. *Survivable Robotics Systems: Reactive and Homeostatic Control*, 135–154. Prentice-Hall, 1993.
4. Baillie, Jean-Christophe and Francois Serra. "Aibo programming using OPEN-R SDK Tutorial". URL $www.ensta.fr/\ baillie/tutorial\_OPENR_ENSTA-1.0.pdf$.
5. Braitenberg, Valentino. *Vehicles: Experiments in Synthetic Psychology.* MIT Press, 1984.
6. Brooks, Rodney A. "A Robust Layered Control System for a Mobile Robot". *IEEE Journal of Robotics and Automation*, 2(1):14–23, March 1986.
7. Brooks, Rodney A. "A Robot that Walks; Emergent Behaviors from a Carefully Evolved Network", February 1989. MIT AI Memo 1091.
8. Desai, Rajiv S., Charles J. Rosenberg, and Joseph L. Jones. "Kaa: an autonomous serpentine robot utilizes behavior control". *Proceedings of the 1995 International Conference on Intelligent Robots and Systems*, 250–255. 1995.
9. Fujita, Masahiro and Koji Kageyama. "An open architecture for robot entertainment". *AGENTS '97: Proceedings of the first international conference on Autonomous agents*, 435–442. ACM, New York, NY, USA, 1997.
10. Gat, Erann. *Three-Layer Architectures*, 195–210. AAAI Press, 1998.
11. Harkins, Richard, Jason Ward, Ravi Vaidyanathan, Alexander S. Boxerbaum, and Roger D. Quinn. "Design of an Autonomous Amphibious Robot for Surf Zone Operations: Part II - Hardware, Control Implementation and Simulation". *Proceedings of the 2005 IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 1465–1470. 2005.
12. Konolige, K., K. Myers, E. Ruspini, and A. Saffiotti. "The Saphira Architecture: A Design for Autonomy". *Journal of Experimental and Theoretical Artificial Intelligence*, (9):215–235, 1997.
13. Lewinger, W.A., M.S. Watson, and R.D. Quinn. "Obstacle Avoidance Behavior for a Biologically-inspired Mobile Robot Using Binaural Ultrasonic Sensor". *Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2006.
14. McCullough, David. "uCLinux for Linux programmers". *Linux J.*, Volume 2004(Issue 123), 2004.

15. Morrey, Jeremy, Bram Lambrecht, Andrew Horchler, Roy E. Ritzmann, and Roger D. Quinn. "Highly mobile and robust small quadruped robots". *International Conference on Intelligent Robots and Systems*, 82–87. 2003.

16. Murphy, Robin R. *Introduction to AI Robotics.* MIT Press, 2000.

17. Nilsson, Nils J. "Shakey the Robot". Technical Note 323, SRI International.

18. Pashenkov, Nikita and Ryuichi Iwamasa. "One-Chip Solution to Intelligent Robot Control: Implementing Hexapod Subsumption Architecture Using a Contemporary Microprocessor". *International Journal of Advanced Robotics Systems*, 1(2):93–98, June 2004.

19. Payton, David W. *Internalized Plans: A Representation for Action Resources*, 89–103. MIT Press, 1990.

20. Porta, Josep M. and Enric Celaya. "Force-Based Control of a Six-Legged Robot on an Abrupt Terrain", October 2000. URL $http://www.ee.pdx.edu/\ mperkows/ML_LAB/Giant\_Hexapod/transm3/icar95.pdf$.

21. Russel, Stuart J. and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson Education, 2003.

22. Saranli, Uluc, Martin Buehler, and Daniel E. Koditschek. "RHex: A Simple and Highly Mobile Hexapod Robot". *International Journal of Robotics Research*, 20(7):616–631, July 2001.

23. Thrun, Sebastian, Wolfram Burgard, and Fox Dieter. *Probabilistic Robotics.* MIT Press, 2005.

24. Ward, Jason. *Design of a Prototype Autonomous Amphibious WHEGS Robot for Surf-Zone Operations.* Master's thesis, Naval Postgraduate School, June 2005.

25. Woolley, Brian and Gilbert Peterson. "Unified Behavior Framework for Reactive Robot Control". *Journal of Intelligent and Robotic Systems.* URL `http://dx.doi.org/10.1007/s10846-008-9299-1`.

26. Xiao, Jizhong, Jun Xiao, Ning Xi, Hans Dulimarta, R. L. Tummala, Mark Minor, and R. Mukherjee. "Modeling, control, and motion planning of a climbing microrobot". *Integrated Computer-Aided Engineering*, 11(4):289–307, December 2004.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)* 01–03–2009	**2. REPORT TYPE** Master's Thesis	**3. DATES COVERED** *(From — To)* May 2007 — Mar 2009

4. TITLE AND SUBTITLE Unified Behavior Framework in an Embedded Robot Controller	**5a. CONTRACT NUMBER** **5b. GRANT NUMBER** **5c. PROGRAM ELEMENT NUMBER**
6. AUTHOR(S) Lin, Stephen S., Capt, USAF	**5d. PROJECT NUMBER** 09–219 **5e. TASK NUMBER** **5f. WORK UNIT NUMBER**
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 Hobson Way WPAFB OH 45433–7765	**8. PERFORMING ORGANIZATION REPORT NUMBER** AFIT/GCE/ENG/09-04
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFRL, Sensors Directorate, Reference Systems Attn: Jacob Campbell Bldg 620, Room 3AJ39 2241 Avionics Circle WPAFB, OH 45433-7333 DSN: 785–6127 x4154 Email: Jacob.Campbell@wpafb.af.mi	**10. SPONSOR/MONITOR'S ACRONYM(S)** AFRL/RYRN **11. SPONSOR/MONITOR'S REPORT NUMBER(S)**

12. DISTRIBUTION / AVAILABILITY STATEMENT

Approval for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Recent technological advances produce small, inexpensive, embedded hardware platforms that are powerful enough to match robots from just a few years ago. The Unified Behavior Framework is a flexible, responsive control architecture that has not been applied on embedded systems in robots. This thesis presents a development of the Unified Behavior Framework on the Mini-$WHEGS^{TM}$, a biologically inspired, embedded robotic platform, which is a small robot that utilize wheel-legs to emulate cockroach walking patterns. Wheel-legs combine wheels and legs for high mobility without the complex control system required for legs. Also included is a color camera and a rotary encoder, enabling the Mini-$WHEGS^{TM}$ to identify color objects and track its position. A hardware abstraction layer designed for the robot in this configuration decouples the control system from the hardware and and provide the interface between the software and the hardware. The result is a highly mobile embedded robot system capable of exchanging behavior modules with much larger robots while requiring little or no change to the modules.

15. SUBJECT TERMS

robotics, artificial intelligence, embedded, microcontroller, biologically inspired

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 Peterson
U	U	U	UU	67	**19b. TELEPHONE NUMBER** *(include area code)* (937) 255–3636 x4281; Gilbert.Peterson afit.edu