Phoenix & Dragon: Parallax BoeBot / Lego Technics Hybrid Solutions to the Reactive Maze Traversal and Deliberative "Cheese" Location Problems

D. Boucugnani
Artificial Intelligence Center

D. Barnhard
Cognitive Science Department

D. Caveney
Computer Science Department

The University of Georgia, Athens, GA
dagustin@uga.edu

Abstract - *Our mission involved building autonomous robots using the Parallax BoeBot kit as a base and making any structural or electronic changes required to complete the task (within reason and ensuring the safety of the BoeBot and its original component parts!) The first task, reactive maze traversal, was completed utilizing two versions of our BoeBot Phoenix, one of which used touch sensors as the sole receptor of information, with the final version utilizing binary IR sensors in lieu of side touch sensors, but still employing a central touch sensor to detect frontal objects. The second task involved devising a solution to the "cheese problem" (i.e., finding a goal in a simple maze, returning to the start position and moving efficiently back to the goal). We devised an absolute positioning scheme in terms of straight-line distance and direction from the goal state. By incorporating minor directional changes to move around obstacles, the BoeBot was able to move back to the vertical coordinate of the goal state (i.e., the start line or the cheese). In short, by utilizing Lego Technics components in addition to various sensors, proper control schemes and world representations, we were able to adequately solve both problems.*

1 Introduction

We designed, constructed and tested an autonomous Parallax BoeBot solution to two problem domains: the task of navigating through a simple maze and the task of sequentially locating a goal (hereafter "cheese"), returning to the start position, and returning to said goal. The maze navigation task is necessarily a subtask of the cheese-location task; therefore, structural and program modifications were made to the first design in order to add the additional functionality required for the final solution.

The maze navigation problem was solved chiefly utilizing a reactive control mechanism with some rudimentary deliberative control. Thus, no internal mapping was used. Instead, successful maze traversal was dependent on sensor state computations (some of which included prior sensor states) in tandem with randomized response.

The specifications of the maze included: walls made from stacked books (conference proceedings), with corridors, corners, traps and exits arranged in novel fashion. The entire maze was placed atop a standard-sized conference table with the robot vehicle (hereafter the Phoenix, "rising from the ashes of our previous attempts") placed randomly in the maze.

The existing BoeBot from the maze traversal problem was then additionally modified and reprogrammed to solve the cheese-location problem. The cheese-location problem is defined by having a search goal (the cheese) placed randomly in a maze similar to that described above. The robot would then search for the cheese, and upon finding it, return to its starting location. Then the robot returns to the cheese in as efficient a manner as possible. Our solution involved engineering a necessarily simple internal representation scheme of the BoeBot's movement domain that could be formed and modified "on the fly". By developing a discrete movement scheme (hereafter "Dragon Movement"), we were able to implement a domain representation using vector mathematics. Indeed, this scheme seems particularly suited to platforms with small memory and processing capabilities in that vertical and horizontal absolute positioning can be achieved in 20 bits. The final prototype (Figure 1.1) was called "the Dragon" due to its "side winding" movement.

To build these autonomous robot vehicles, we used a standard Parallax BoeBot constructed via kit from a Parallax

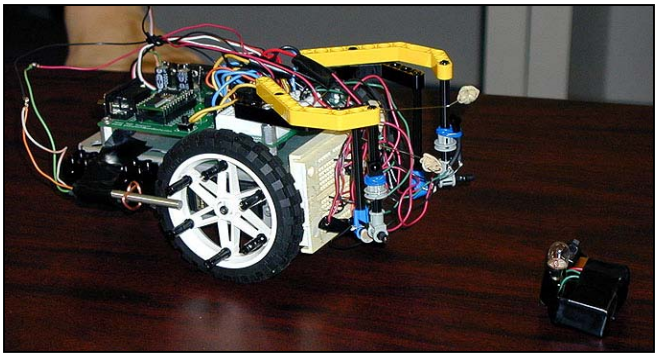

Figure 1.1 – The Dragon and the Cheese

Basic Stamp 2, a Parallax Board of Education (Revision B), and a Parallax BoeBot chassis and servo kit. In this report, we will describe structural and electronic modifications as well as signal processing and computation strategies used in the development of the two robot vehicles. Finally, the fusion of all the previously mentioned modifications and strategies will be discussed using our PBASIC (Parallax Basic) code sources as models. The source code is available upon request and will be available on the web at http://ais.ai.uga.edu/boucugnani/robotics/src.html.

2 Structural and Electrical Modifications - Phoenix

Our BoeBot underwent a series of structural modifications in an attempt to find a combination of sensor mountings that would be flexible, yet sturdy and low-profile. We experimented with modification of the whiskers, addition of an auxiliary breadboard, addition of Lego Technic parts to the BoeBot chassis to support rapid development of new sensor ideas and mountings, and the development of a front and back sensor made entirely from Lego Technic parts. Using combinations of the above modifications, we developed three working versions of the Phoenix with differing maze results.

2.1 Whisker Modification

Our first attempt involved building a standard BoeBot with a two-bit whisker touch sensor (hence two autonomous whisker assemblies). To our chagrin, when the BoeBot was tested in a simple maze, the whisker attachments would oftentimes fail in one of two ways: (1) a whisker would embed itself between books in a stack or slip on a surface that should have registered as an obstacle; and (2) whiskers would sometimes bend in the wrong direction.

To alleviate the first problem, we devised a gripping mechanism for the ends of the whiskers to ensure that they would not slip or embed themselves within obstacles (see Fig. 2.1). This mechanism consisted of a rubber band looped around the ends of each whisker, which is superior to dipping each end in rubber cement (as suggested by Parallax) due to the additional ability of the whiskers not to become embedded in obstacles (e.g. between book pages). A surprising benefit was that the whiskers had, in our opinion, much more sensitivity due to the supplementary gripping power.

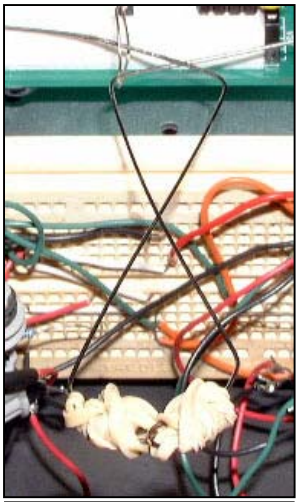

Figure 2.1 – Rubber Band Gripping Ends for Whiskers

A second whisker problem was noted: the original whisker switch mechanism was simply the whisker impacting the first pin of a three-pin header on the Board of Education breadboard (see Figure 3.3 in Robotics! Version 1.4). This proved inadequate for our purposes and in light of a turning robot. If the robot were to turn (e.g. a u-turn) with an obstacle hitting a whisker from behind, the whisker would be bent in an improper direction and never impact the three-pin header, thus risking damage to the whisker assembly and improper sensory response to an obstacle. Thus, we devised an omni-directional switch (see Figure 2.2) for each whisker by forming a length of non-stranded, semi-rigid wire into a loop and soldering the loop to the first pin of the three-pin header, passing each whisker through its respective loop. Where each whisker passed through the other whisker's loop, a band of electrical tape was applied (to the whisker) to prevent a mistaken touch (where one whisker would touch the other whisker's loop and thereby cause the wrong signal to be sent). This solution was effective and appropriate to a touch-sensor only, maze-traversing BoeBot, which, when properly programmed, will eventually traverse simple mazes.

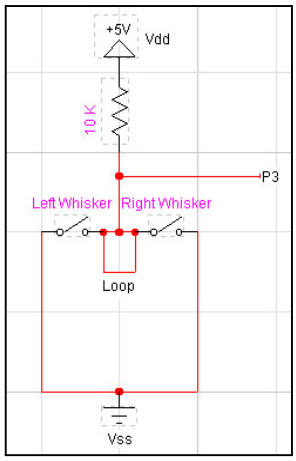

Figure 2.2

In the final version of the Phoenix, we decided to use binary infrared sensors as our primary forward obstacle detection system. However, with the two infrared sensors mounted at angles in the front of the BoeBot, a blind spot occurred between the sensors, so that the BoeBot could not detect obstacles directly ahead. To alleviate this problem, we developed a 1-bit whisker touch sensor (two whiskers but with one loop in which both whiskers pass through. The two whiskers were arranged so that their tips would protrude between the two infrared sensors, thus allowing the BoeBot to detect a physical, dead ahead obstacle that the infrared sensors had missed. This configuration was possible in light of the fact that each whisker originates from a Board of Education attachment point that is wired to the same terminal (i.e. Vss or ground) (see Fig. 2.2).

2.2 Front Breadboard Attachment

An additional breadboard was mounted to the front of the BoeBot chassis to provide additional connection points and room for additional circuitry design. Originally, the binary IR sensors and receivers were mounted directly to this breadboard. This did not prove to be a sound solution as failure to sense obstacles in the front caused damage to the sensors and/or sensor – circuitry disconnection. A more robust solution is described below.

2.3 Lego Technic Chassis Modification

In light of the difficulty in adding third-party components to the chassis of the BoeBot, we decided to

utilize the Lego Technics system for rapid prototyping of structural additions. Our first attempt involved the addition of two nine-hole Lego beams to the rear of the chassis. This allowed for the attachment of a novel Lego Rear Touch Sensor assembly as well as a tower for a Sharp GP2D02 IR range sensor. This assembly was later reduced to remove the Lego Rear Touch Sensor (as it was too bulky for tight maze turns) but to maintain the tower for the IR range sensor. In addition, two Lego black 7-hole bent beams were mounted to the front of the BoeBot (where the original whiskers were attached) and binary IR sensor/receiver combinations "hung" from each bent beam using additional Lego parts. This configuration (which we term a Technic IR Post) allows for the easy, manual movement of the binary IR sensor assemblies to arrangements conducive for the sensing task at hand. When combined with the one-bit whisker touch sensor assembly described in §2.1, this provided an effective solution to the mechanics of forward obstacle detection.

Lego parts are light, provide a convenient way of fast prototyping design ideas and can solve many mechanical issues without the builder needing to digress into the purely mechanical engineering issues of designing and machining custom parts. However, one must take care, especially when using Legos, to keep the bulkiness of the design to a minimum. Failure to do so can affect the turning radius or the width of the BoeBot adversely. Additionally, it must be noted that Lego creations tend to be on the fragile side; however, reinforcement techniques involving crossbeams significantly improve robustness.

2.4 Quick and Easy Voltage Regulation

Some electronic modifications to the kit were performed. A simple voltage regulation scheme is to use the 5.0V regulated voltage already provided by the Board of Education. Under this voltage, the servos seem to run very smoothly and can still be regulated by Pulse Width Modulation (PWM). By abandoning the unregulated servo ports on the Board of Education (Rev 2), constructing our own ports from Vdd (regulated) power instead of Vin (unregulated) power and rerouting the P12 and P13 signals, we were able to provide constant power to each servo (until the Vin voltage dropped to under 5.0V, whereby the onboard voltage regulator ceased operation). Fig. 2.3 illustrates a circuit diagram of this solution.

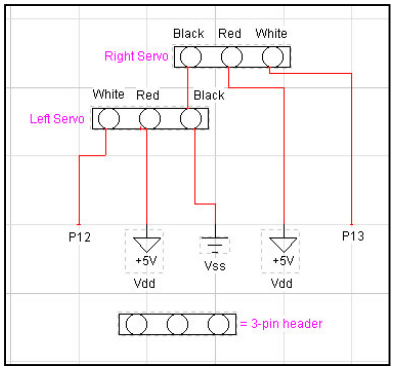

Figure 2.3 – 5.0V Regulation

3 Structural Modifications – Dragon

3.1 Touch Sensor / Wheel Projection Combination Sensing

It has long been established that Lego Technics parts are perfect for rapid prototyping of new robotic sensors. Hence, we utilized two large Lego Technics wheels (white) with long black connectors inserted into each radial socket, and a spring-and-loop touch sensor. This assembly is pictured in Figure 3.1. Note that the touch sensor is mounted using

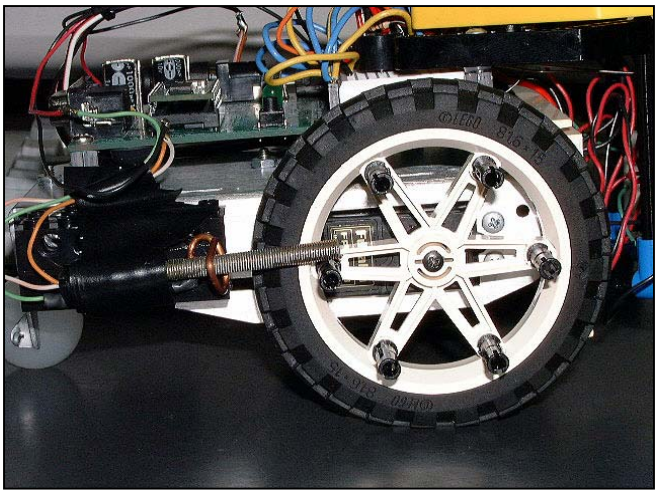

Figure 3.1 – Touch Sensor / Wheel Projection Combination assembly

electric tape – this was for rapid prototyping. A more robust solution would be to use screws and crimps so that, over time, the touch sensor would not become dislodged from the frame. The connection to the Board of Education is exactly the same as the standard touch sensor circuit described in the Parallax Robotics manual. In theory, the idea is simple. As the wheel turns, the spring on the touch sensor moves to contact the loop. After the peg has been passed, the spring moves back into place, awaiting the next peg to impact. It is true that this scheme cannot, by itself, differentiate between forward and reverse movement; however, in conjunction with the pulsout values provided to the servos, the control program was able to differentiate and thus keep track of how far each wheel had turned regardless of direction. Indeed, this method gave us 60-degree resolution of wheel movement, sufficient for our purposes.

3.2 "Komodo Dragon" Movement

When we attempted to keep track of absolute movement by turning both wheels at the same time, invariably we would veer off straight-line motion in one direction or the other. Indeed, we tried many different control programs and schemes to correct for this problem, but we still could not yield perfect results. Finally, an epiphany: why not move one wheel at a time, causing the BoeBot to amble much like we had seen a Komodo Dragon do? Willing to try anything at this point, we wrote a quick control program to move the BoeBot forward ten steps utilizing this type of motion. To our surprise, the BoeBot moved in a perfectly straight line,

provided that the touch sensor springs were placed at equivalent locations with respect to each other. Using this scheme, the BoeBot would move in a straight line diagonal to its original position. By moving one spring to touch a peg and the other to a central location between two pegs, we were able to correct for this diagonal motion and cause the BoeBot to move in a straight, parallel line to its original position and orientation! This breakthrough solved our problem of keeping track of absolute position as well as the nagging problem of straight-line motion.

3.3 A "Sense"-able Way to Detect the Cheese

For our cheese, we decided to go with a light source connected to a nine-volt battery and a switch. This provided sufficiently bright light to differentiate the cheese from ambient signals (i.e. overhead fluorescent light and glare from the surface being traveled upon). To detect the cheese, we used photoresistors attached to the Board of Education using the same circuit as described in the Parallax Robotics manual. These photoresistors were mounted below the infrared sensor as is shown in Figure 3.2. In the main program, these photoresistors were turned into virtual boolean sensors by utilizing threshold values. These thresholds were determined by placing the light source (i.e. the cheese) at an approximate BoeBot chassis length away from each photoresistor, whereby the capacitor discharge time was recorded using the debug window of the Basic Stamp IDE for each resistor. Therefore, we were able to use photoresistors without violating the constraints of the challenge. This design would be easy to modify to produce a light-tracking or light-repulsed robot.

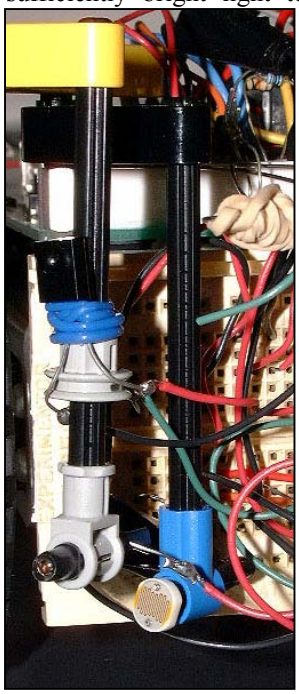

Figure 3.2 – Photoresistor / IR Hanger

4 Computational Strategies - Phoenix

Code for the first version (phoenix1.bs2) and the final version (phoenix2.bs2) are provided on the web. In the following sections, we provide an overview as to the strategies represented in each program.

4.1 Whisker Touch Sensor Phoenix

This program (phoenix1.bs2) is a highly-modified version of Program Listing 3.3 in the Parallax Robotics! Version 1.4 Student Workbook. The gist of the program is that the BoeBot will maneuver through a maze utilizing only touch sensors, and will remember information from previous states to help determine what action to take in the current state. First, the program checks the state of the whisker sensors and stores this value in variable *state* having size nib (4 bits). It then branches to one of four different subroutines depending on the value of *state*. If both touch sensors are activated at the current state, branch *state* (00) dictates that the BoeBot will perform a u-turn due to the perception that a wall is directly ahead. This proceeds in a similar fashion for each combination of *state*. The program also keeps count of how many times it has perceived a left or a right touch sense. If the BoeBot has not moved forward a set amount and both touch sense variables are above a certain threshold, the BoeBot will perceive that it is trapped in a corner and will execute a u-turn. The u-turn has some interesting properties. First, it causes the BoeBot to back up a bit (to preserve turning radius and to protect the whiskers). Then, it executes a turn of random length (e.g. in one instance it may turn 90 degrees, in another instance, it may turn 160 degrees). This ensures that the BoeBot will not get caught in an operational loop. This simple program is what we used for the first maze competition.

4.2 Binary IR Sensor / Touch Sensor Hybrid Phoenix

This program (phoenix2.bs2) is a highly modified version of Program Listing 5.3 in the Parallax Robotics! Version 1.4 Student Workbook. In this program, we attempted to implement a subroutine that remembers the previous state of the IR sensors, in order to detect when a "hole" in a wall was being passed (presumably to turn towards the hole, a semi-deliberative response). The binary IR sensors operate in a similar way to the whisker touch sensors except that when both are registering an object and the 1-bit touch sensor does not perceive an object, the BoeBot will continue forward (it presumes that it is in a corridor and needs to investigate it to the end). If the front touch sensor strikes an object, the response is dependent on the state of the IR sensors. If the IR sensors do not detect an object to the left or right, the BoeBot will turn in a random direction (it perceives a wall in front). If they detect an object to either the left or right, the BoeBot will turn in the opposite direction. If they are both detecting an object, then the BoeBot perceives that it is in a corner and executes its corner exit routine.

4.3 Binary IR Sensor / Touch Sensor Hybrid Phoenix with Bi-Level Distance Sweeping

This program (phoenix3.bs2) is a modified version of the previously-described program (phoenix2.bs2) that utilizes IR distance sweeping (first attempting to detect an object close up and then attempting to detect a far away object if the near range detection did not detect an object). It works in a manner similar to phoenix2.bs2 with the exception that it uses both a short-range and a long-range IR distance detection scheme to make decisions as to the next move. For example, if an object is detected in the distance in one direction, the BoeBot might turn in the direction of that object to investigate. If objects are detected in the

distance on both sides, the BoeBot may perceive that it is in a wide corridor and monitor the long distance sweep to detect a break in the wall unperceivable by the short sweep. It also utilizes a routine called *smart_turn*, which uses information from both sensor sweeps to decide which way to turn if the front touch sensor is activated (as opposed to turning in a random direction).

4.4 Comparision with Pioneering Robots

W. Grey Walter was a prominent, British educated neurophysiologist doing some of his most interesting work in the 1950's: the design and testing of his Machina speculatrix tortoise. Indeed, his original intent was not to be at the forefront of autonomous robotics research. According to Owen Holland of the University of the West of England, Bristol: "He wanted to prove that rich connections between a small number of brain cells produces very rich behavior," (Robot Critters, New Scientist Web Article), in order to aid science in ascertaining the nature of human cognition.

Walter's turtles were autonomous robots powered by a two-vacuum tube computer made completely from 1950's electronic components. The M. speculatrix had a rotating photocell and a tricycle drive system (unlike the BoeBots, which have a dual-front wheel drive with a caster in the back), allowing the robot to search for and react to differing light levels by movement towards or away from the light source. Undeniably, M. speculatrix was a completely reactionary entity – it had no deliberative (planning) behavior whatsoever. This closely matches the majority of our BoeBot mechanics and behavior with the exception that: 1) the BoeBots had several, different types of sensors such as two levels of binary infrared, a ranging IR sensor, touch, and in one experiment, actual photocells (not reported in this paper) and 2) the third BoeBot began to do some internal processing in light of several different environmental signals and had a little bit of knowledge about the world in the form of storage of previous states. M. speculatrix had a single touch sensor which can best be described as a hanging "shell" over the robot body which, when impacted on any side, would trigger the sensor switch (one response to a collision with an obstacle). However, the majority of its interesting behavior was derived from its light sensing ability.

M. speculatrix had four internal states: search, move, dazzle and touch. Dazzle was a state where the robot encountered light that exceeded a brightness threshold, while touch was a state oscillating between dazzle and search. Our final BoeBot had several internal states corresponding to each combination of sensory input.

The Braitenberg vehicles were very similar to Walter's turtles. They were based on cybernetic principles whereby complex autonomous vehicles can be built using simple electronic and mechanical components. These constructs were completely imaginary and contained in his 1984 book *Vehicles*. Again, Braitenberg's main focus was not to advance the field of robotics; rather, he was focusing on the psychology of emergent complex behavior from a synthesis of simple components. Indeed, the BoeBot could be considered a collection of simple components (in todays terms) that can operate together to produce increasingly complex behavior. In this way, the BoeBots and Walter's turtles perpetuate the idea of constructing intelligent behavior purported by Braitenberg.

5 Computational Strategies - Dragon

5.1 Keeping Track of Distance and Direction Using Vectors

Physics and trigonometry always have ways of slipping into normal, everyday life. Even with our simple BoeBot design, we found a use for the trigonometry that has dogged high school students for years. Indeed, our most major breakthrough was purely accidental – we discovered that a white Lego wheel, when attached to the side of a BoeBot and turned alone (i.e. with the other wheel held stationary), the BoeBot will execute a nearly perfect 360-degree turn in exactly 16 clicks (touch sensor firings)! This enabled us to develop a heading system enabling us to keep track of the BoeBot's current direction. This, coupled with the counting of steps forward and backward (1 click of each wheel in the same direction), supported the formation of a vector system of position, in which the distance from the start point (vertical in this case, but which could be extended to both vertical and horizontal) can be computed. By keeping track of the number of steps the robot has proceeded in a certain direction, vertical (and horizontal) vector components of position can be computed.

Hence, at each direction change, the previous direction and steps in that direction are converted into a vertical component and then added to, or subtracted from, the vertical_steps_from_home global variable. Table 5.1 illustrates how the vertical components of the distance from origin are computed in the program. The subroutine vert_step_bearing_calc, performs these calculations every time a directional change is performed.

Bearing	Calculation
1:	increment horizontal step total by full step_count
2,16:	increment horizontal step total by step_count * cos 22.5 * [(924 * step_count) // 1000]
3,15:	increment horizontal step total by step_count * sin 45.0 * [(707 * step_count) // 1000]
4,14:	increment horizontal step total by step_count * sin 22.5 * [(383 * step_count) // 1000]
5,13:	no increment
6,12:	decrement horizontal step total by step_count * sin 22.5 * [(383 * step_count) // 1000]
7,11:	decrement horizontal step total by step_count * sin 45.0 * [(707 *step_count) // 1000]
8,10:	decrement horizontal step total by step_count * cos 22.5 * [(924 * step_count) // 1000]
9:	decrement horizontal step total by full step_count

Table 5.1 – Vertical component calculations

5.2 Source Code Discussion

The source code (dragon.bs2) is, admittedly, not quite complete. However, it seems to do an acceptable job at finding the cheese (as specified), returning to the goal, and finding the cheese again with minimal obstacles. Again, the robot's absolute position is stored in two variables, goal_bearing (4-bit nibble)
vert_steps_from_home (8-bit byte), which is a representation of how many steps the BoeBot would have to take along goal_bearing to find its way to the vertical location of home.

The source code is split into five behavior subroutines: seek_cheese, cheese_found, return_to_goal, goal_found, and return_to_cheese. The seek_cheese subroutine loops on itself until the cheese is encountered (via one of the photosensor values being less than the threshold for that particular photosensor). This state causes the loop to terminate, whereby the program goes on to the next behavior, cheese_found. The subroutine avoid_obstacles is the main navigation program – it is here that, from the states of the sensors, the next movement decision is made. Normally, the BoeBot will simply step forward one step. However, depending on the location of the obstacle, the BoeBot will make standard obstacle avoidance movements. The difference here is that the BoeBot will also keep track of how it has moved to avoid the obstacle and use this new directional information, along with the previous directional information in modifying vertical_steps_from_home to represent the new location state. For now, the BoeBot simply moves in a random fashion – it does not have a set search pattern. Indeed, this is where the code could be most improved on.

When the BoeBot has found the cheese, it enters the cheese_found subroutine. This causes the program to calculate the final vertical_steps_from_home value and point in the direction of the home location. The program then enters the return_to_goal subroutine, whereby it attempts to move along the goal_bearing bearing while decrementing the value of vertical_steps_from_home. This works in much the same way as seek_cheese in that it reuses the navigation subroutine avoid_obstacles. An interesting point to note is the use of a fuzzy condition – when the robot has reached a certain, predefined vertical_steps_from_home value and then encounters an obstacle, the program assumes that this is the back wall of the search space and thus that the robot is in the goal area.

After returning to the goal area, the program executes the goal_found subroutine. Here, the robot simply points in the vertical direction of the cheese, which is always in the direction opposite to goal_bearing. Finally, the program executes the final subroutine, return_to_cheese, which begins just like the return_to_goal subroutine except that its new vertical position goal is cheese_vert_location, which was set when the cheese was found. After returning to the vertical location of the cheese, the robot then performs a horizontal sweep for the cheese, using its photosensors to determine whether or not the cheese's horizontal location has been reached. The program then ends with an audible signal.

6 Synthesis and Fusion

We were able to effectively combine software with hardware using simple signal processing techniques. Considering that most of our sensors were binary in nature, the signal processing was somewhat straightforward. One of the challenges we did not fully investigate was including distance sweeping for the binary IR sensors. We will most definitely further our research into this area, as well as fully integrating the Sharp GP2D02 into any further projects. We achieved good synthesis of signal processing and software in that the software properly handled the signals provided by the various sensors in the majority of cases. Of course, there will always be "holes" in perception (i.e. perception of the outside world can never be complete), but perhaps with the addition of more sensors and more robust software, the response of the BoeBot will be more appropriate to various situations. The bridge between raw sensor output and the Basic Stamp inputs was handled by electronic modification schemes, most of which were predefined in the Parallax Student Workbook.

The cheese-location challenge indeed tested and approached the limits of the Parallax Basic Stamp 2. We filled the EEPROM and most of the registers with variable and program data. The robot was able to complete basic cheese finding and returning exercises, although if we had spent less time on the absolute positioning problem, we might have had a more refined program. Due to the nature of the PBASIC programming language (as compared to C, for example), coding for this project was somewhat tedious. Perhaps by abandoning the Basic Stamp in lieu of a micro-controller utilizing a C-compiler, more could be done in less time, while still keeping the same chassis, servos and sensors.

Our future directions will most likely focus on using a much more powerful platform, perhaps by developing a handheld or laptop-based robot. In this event, faster and more robust sensors will need to be developed or discovered. It might be interesting to have a "mother" robot communicating with and controlling several "worker" BoeBots, if we can find or develop a convenient communication protocol

7 References

Hogg, Martin, Resnick. "Braitenberg Creatures" MIT Media Laboratory. (1991) http://fredm.www.media.mit.edu/people/fredm/papers/vehicles/vehicles.html

New Scientist Magazine. "Robot Critters". Robots in the News. http://www.robotbooks.com/robot-critters.htm

Parallax, Inc. "Robotics Student Workbook Version 1.4" Parallax, Inc., 2000.

Complete paper and source code available online at http://ais.ai.uga.edu/boucugnani/robotics/src.html.