

Design and Implementation of a 5-Sonar Maze Running Robot

Adrien Grise (agrise@cs.uml.edu)
Tom Kneeland (tkneelan@cs.uml.edu)

May 22, 2003

Abstract

When the term “robot,” is mentioned, most people think of an autonomous device like those seen in science-fiction movies. Mobile, intelligent, and self-reliant, these fictional devices are still well off in the future, however, smaller, simpler devices can and are being developed locally to study some of the basic issues involved with higher-intelligence devices. One such device is an autonomous device capable of navigating its way through a maze. Given basic off-the-shelf resources, such as sonars, Cricket devices, and LogoChip ICs, a small mobile robot can relatively easily be developed to accomplish the traversal of a maze.

Introduction

When we signed up for the University of Massachusetts Robotics course we pretty much knew independently that if a project were required our first choice would be a mobile device such as a maze-navigating robot. After all, if asked what a robot is, the average person immediately thinks of an autonomous, “classic,, robot. To make our lives easier there was already a maze built in the university robotics lab for us to experiment with.

We started by trying to devise a set of goals we wanted to achieve with our robot. Of course this was long before we really had an understanding of the types of challenges we would face when actually implementing our design. For starters, we not only wanted our robot to be able to navigate the maze, we wanted it to be able to do it in reverse. That is, we wanted it to remember how it got to a certain point in the maze, and then turn around and retrace its route to get out. This feature led us to think of our robot as being able to perform a search and rescue function. It would move around the maze looking for some token, and once it was located, the robot would be able to turn around and exit the maze by the way it came in.

Early in the class we were introduced us to the Handy Cricket². The Cricket is a small programmable device consisting of two standard motor outputs, two sensor inputs, an IR programming interface and a two-wire bus port. The whole system is controlled by a PIC micro-controller running a Logo virtual machine created by the system’s designers at MIT. One of the intended uses of the cricket was to facilitate devices that could communicate and coordinate with each other, hence the name Cricket. We thought we could put that to use by having our robot, upon exiting the maze, tell another robot how to find the token. We thought we could even devise a system that could analyze the route taken by the robot and try to optimize it.

Some of these ideas turned out to be a bit ambitious, and were unfortunately left out of the final implementation. Even with the removal of ideas like searching for a

token and logging the path used to locate it, there were still a host of challenges to solve. One of the inspirations that ultimately led to our basic design was a maze robot described by Mike Linnen we found on the Internet¹. On the web page he describes a robot that uses a couple of side mounted infrared range sensors to navigate a maze by following a right hand or left hand wall. It was a design he had a fair amount of success with, even though it did not have any forward facing sensors and would have to bump into a wall before realizing it had to turn. Our design sought to solve this problem by using forward facing sensors as well as side-mounted sensors.

We tried several prototypes before settling on our tread-based system. Several ideas were tested only briefly, while others remained in the system till further development and testing noted the deficiencies. Ultimately, the robot structure was settled on and development could continue to the more advanced design issues required to complete the project. Given this appropriate platform, and the desire to make the robot smart enough to not have to bump its nose into a wall before turning, we set about implementing our Cricket, LogoChip, and Sonar Sensing Independent Craft (a “CLASSIC,, robot) described in the remainder of this document.

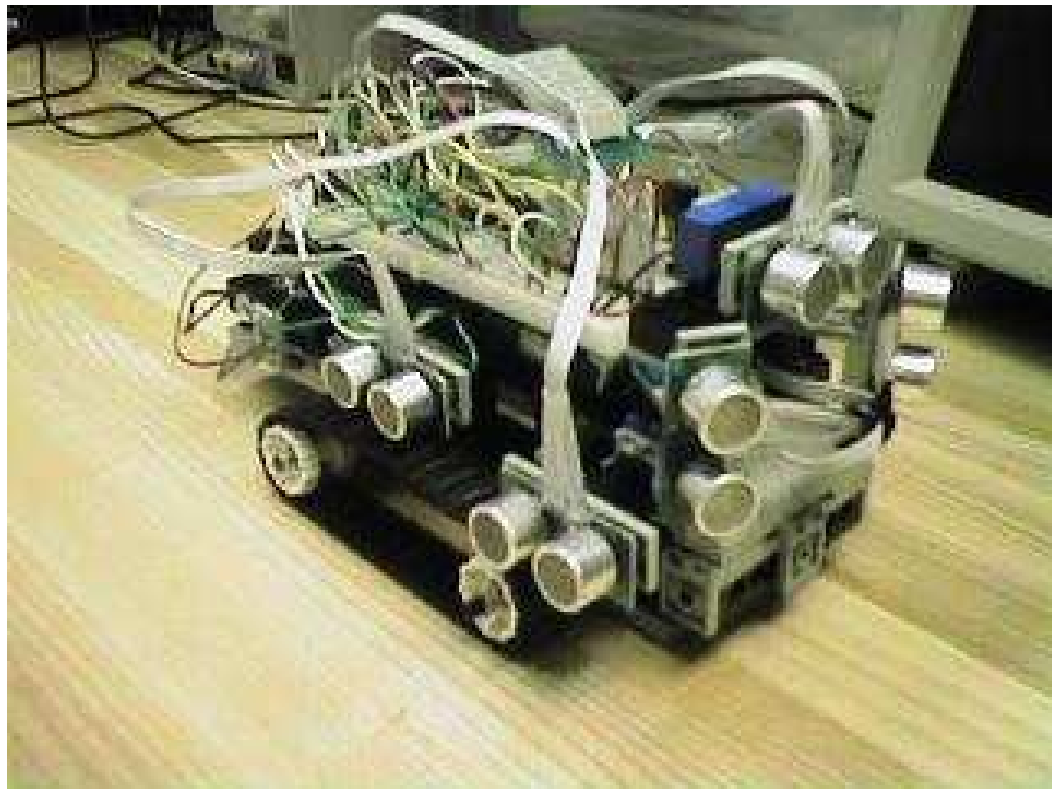


Figure 1: The sonar-based mobile dual-treaded robot

Pent-Sonar Bus Device

In order to develop a robot that used sonar to navigate a maze we had to develop a way to interface the Handy Cricket with not just one sonar device, but with at least four sonar's and ultimately five sensors. The specific sensor we used was the Devantech

SRF04 Ultrasonic Range Finder⁴. It uses a standard TTL 5v interface and can detect objects between 3cm and 3m away with a +/- 3cm resolution. Unlike the SRF08 Ultrasonic Range Finder, the SRF04 requires the host to perform the necessary timing and distance calculations.

A Cricket Bus device implemented using a Logo Chip³ seemed to be the perfect solution for our interface issue. The Cricket Bus is a simple one wire serial bus where all the low-level communications are handled by the Logo Virtual Machine. The Logo Chip also provides numerous programmable I/O pins; more than enough to coordinate multiple sonar devices.

Through some prototyping it was discovered that the sonar sensors had trouble with angles greater than 45 degrees. When our robot moved toward a wall at a 45-degree angle, the distance reported was much greater than the actual distance. We could only hypothesize that this was caused by the sonar signal reflecting off the wall, and causing an abnormally long delay before the return 'ping' was detected. Our solution to this problem was to add two additional sonar sensors to the front of the device, mounted 45-degrees apart on either side of the original sensor. Some recent research on the Internet revealed that the optimal distance between sensors is 20 degrees, so we may have been able to modify our design to place only two sensors on the front of the robot, 20 degrees apart.

It was easy to figure out how the sonar devices worked. We started by studying a single sonar bus device implemented by a classmate for an assignment. His sample code showed the polling technique used to determine the time it took for a sonar device to detect its "ping,,. The distance, however, was encoded using a coarse near/middle/far metric, not nearly accurate enough for our navigation.

Each sonar sensor has four pins that need to be connected in order to operate. Two are simply +5v power and ground. The other two can be thought of as data pins. The device's input pin, the Trigger, is used by the Logo Chip to tell the sensor to generate a ping. The output pin, or Echo, is used to measure the elapsed time from when the ping is generated to when the echo is received. When the device generates a ping it sets the Echo pin to +5v which is read as a boolean 1 by the Logo Chip. When the echo is "heard,, by the sonar the Echo pin is cleared, read as a boolean 0 by the Logo Chip. The duration the Echo pin reads 1 is equal to the round-trip time (in milliseconds) the sound traveled from the sonar sensor to an object and back again. We derive a distance, in centimeters, by dividing this time in half and multiplying by the number of centimeters sound travels in 1 millisecond. Due to the latency of our timing loop and the granularity of the sonar sensors, our distance values are limited to multiples of 3.

As it turns out it is possible for a sonar sensor to break. One of the original sensors we tried to use generated a much softer ping than to other sensors and would never hear its echo. This resulted in the sensor always reporting its maximum distance value. We simply replaced it with a working sensor.

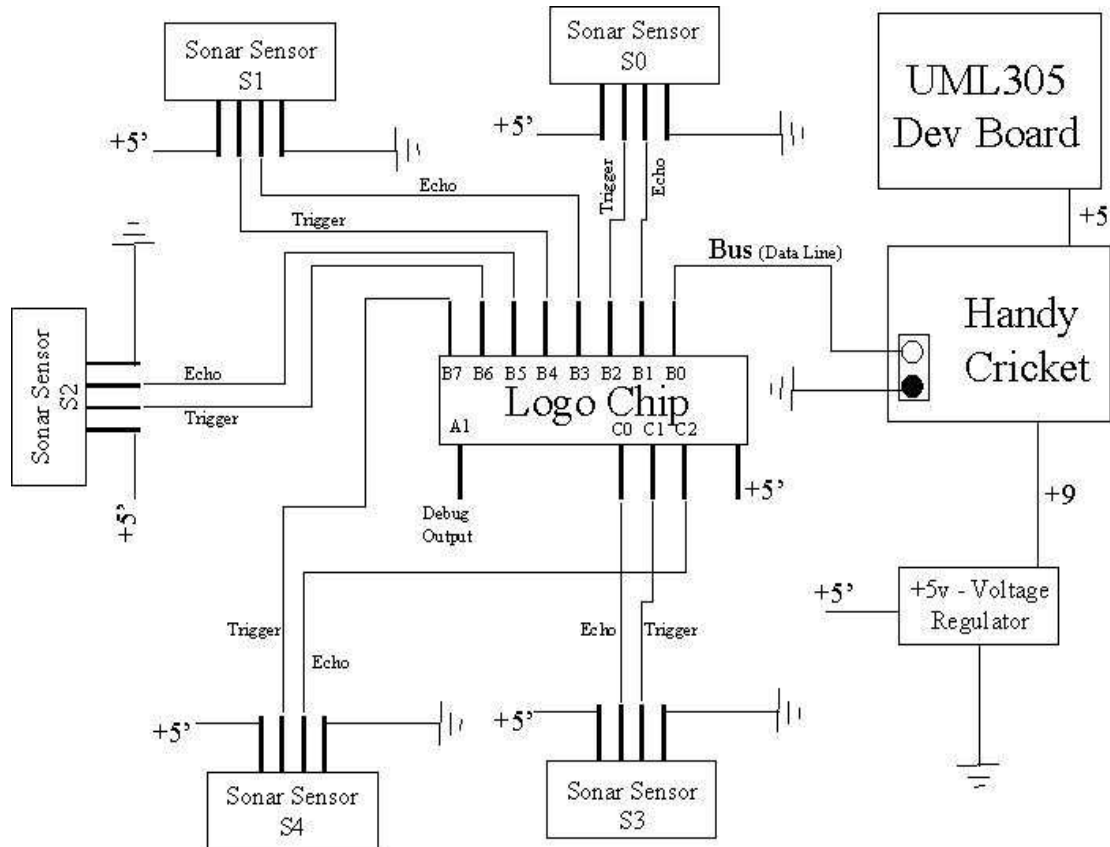


Figure 2: Schematic diagram of mobile robot circuitry

Cricket Bus Communications

The Cricket Bus protocol was kept simple. The Handy Cricket makes requests for specific sonar values (S0 through S4) and the Sonar Bus Device provides the latest distance value it has for that sonar. The sonar device constantly scans the sonar sensors in a round robin fashion so it always has a value available for the Handy Cricket.

The bus device software is written entirely in Logo. Although Logo is not a task based system the software was written with two task priorities in mind, high and low. Our first priority was to the Handy Cricket controlling the robot. We did not want it to have to wait an inordinate amount of time for a requested distance. To achieve this we always check for a bus request at the top of our main bus control loop. When a request is received we return the last sonar reading for the requested sensor from the appropriate global variable. We do not make the Handy Cricket wait while a measurement is taken. Also, during each iteration of the main control loop, we only measure the distance of one sonar sensor. Measuring all five sensors in succession could lead to an unacceptable delay on the Handy Cricket.

Another detail of the implementation that is probably obvious but is worth noting is that only one sonar sensor is active at a time. Aside from being the only way to accurately watch the transitions of an Echo pin it also means that the power draw from the inactive sensors should be minimal, and there is no chance of one sonar ping interfering with another sonar's.

By storing each sonar sensor's distance in a global variable the bus device can operate independent of the Handy Cricket. There is no need to try and anticipate which sonar sensor the Handy Cricket will request next. We have also parameterized the functions responsible for controlling the sonar sensors. This allowed us to define a collection of constants to indicate which Logo Chip port and pins were assigned to each sensor. As a result we have complete flexibility over how the sensors are connected to the Logo Chip. This came in particularly handy when one sonar sensor had to be connected to both Ports B and C.

We did encounter an interesting problem with our bus device in relation to the overhead within the virtual machine associated with processing the Cricket Bus. Frequently the bus device would "hang,, when the robot was close to a wall or object. Bench-top testing revealed this condition would only occur when the Handy Cricket was running, and using the bus. The problem seemed to be exacerbated by the use of the 4-character LED, which also used the bus. A little bit of debugging using one of Port A's output pins revealed that our Logo program was stuck waiting for the Echo pulse to transition from low to high (0 to 1). The original hypothesis was that there was something wrong with the sonar sensor, causing the transition to not occur. This theory however did not explain why it only occurred when the Handy Cricket was running. Upon closer inspection we were able to determine that the pulse was actually occurring but it was occurring so fast that it was missed. This scenario occurs when the Cricket Bus latency is high (i.e. the Handy Cricket is requesting a sonar reading) and an object is very close to a sonar sensor. Under these conditions the minimum width of the Echo pulse is actually around 100us while the bus handling code in the Logo Virtual Machine requires around 200us. There is more than enough room for these two events to overlap, causing the echo pulse to be missed. The solution to this situation was to implement a time out mechanism while waiting for the echo pulse to go high. The timeout we implemented was actually overly generous. We wait the maximum pulse duration of 36ms, as opposed the maximum duration between our Trigger pulse and the time until the pings are generated. This is definitely a good case for optimization.

To finish our discussion of the bus device, we turn our attention to debugging. Debugging an embedded system is always a challenge, usually due to its lack of a primary display. We utilized several tricks to work around this limitation. First and foremost we made use of the Logo Chip's ability to transmit numbers over its serial connection to the host PC. This allowed us to verify the distances returned by each sonar sensor, ensure that the sensors were being polled in a round robin fashion, and lastly ensure proper recognition of Handy Cricket requests over the Cricket Bus.

We also made use of an available bus pin, in our case on Port A. We configured the pin to operate in output mode and then would set its value to "1,, when we entered a "critical,, portion of code, and then set it to "0,, upon leaving. This allowed us to track the software's progress on an oscilloscope. Changing the values of the data pin has much less impact on the performance of the system than did writing to the serial port. This was actually the method we used to detect where the device was hanging while debugging the bus latency issue. Using the oscilloscope to view the data output on the pin saved us the effort of trying the wire up an LED and resistor to our already over-crowded breadboard, plus afforded a higher resolution view of program execution timing.

The Handy Cricket Navigation Software

The Handy Cricket was used to control the navigation of the robot as it moved through the maze. Through the use of a state machine, the Cricket would analyze sonar values and determine the best course of action to take given its current environment. The Cricket would loop continuously, polling all the sonar's, and then enter the appropriate state depending on those values. Initially, the robot would start up in a normal, forward-moving state (state 0 according to the 4-character LED display attached to the robot). From then on, the navigation system would jump to different states as the robot encountered different maze layouts. The basis of our testing and experimentation was a maze built to reproduce the Trinity Robotics Contest maze.

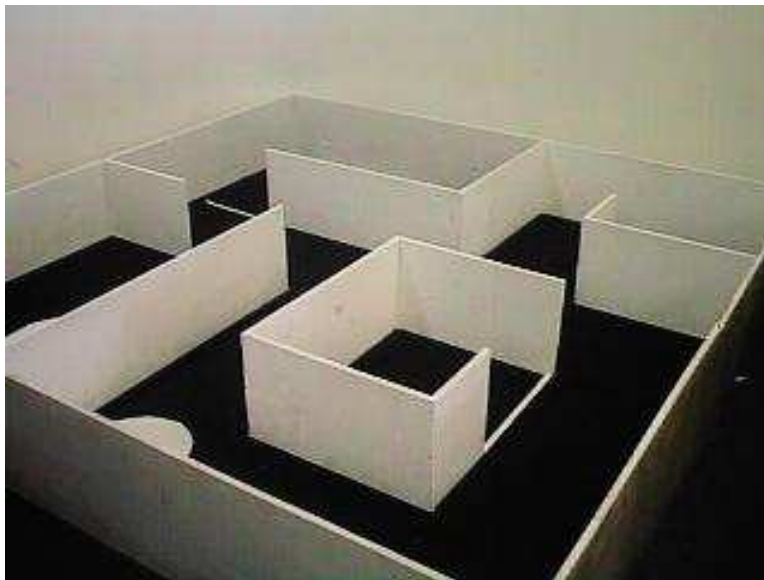


Figure 3: UMass Lowell CS Department robotics maze

The initial state (state 0) had both right and left motors moving the robot forward at full speed (a value of 8 out of a possible 8 for the Lego-based motors). As the robot moves forward, the robot analyzes the two side sonars. If the first sonar is 9 centimeters further away from the wall than the second sonar, the robot has detected that it is veering away from the right wall and enters state 1. If the two sonar readings are roughly equal (i.e. less than the 9 centimeter difference noted previously), the front side sonar is checked to see if it is less than 9 centimeters away from the wall, and if so, enters state 2 to force the robot away from wall. If the front side sonar is greater than 15 centimeters away, the robot knows it is too far from the right wall and enters state 1 as well.

If while we are in state 1, we detect that the second side sonar reports a value greater than 12, we know that there is a large open space to the right detected by both side sonars and we enter state 3. Otherwise, we will stay in state 1, moving forward as we did in state 0.

In state 2, we adjust the power ratios to the motors and cause the robot to turn left. We will continue turning left until the front side sonar reads a distance greater than 9 centimeters and, if so, return to state 0.

In state 3, we adjust the motors' power ratios to cause a right hand turn. If the either of the side sonar distances were read as less than 9 centimeters, we return to state 0.

The frontal collision warning state (state 5) would be encountered if any object were detected within 12 centimeters of the front of the robot. The 45-degree angled sonars also check to make sure there were no objects within 12 centimeters as well and created a collision-avoidance zone. If an object were detected, the Cricket would power the left motor in reverse at power 4 to spin the robot until all three front sonars indicated nothing was within their collision-avoidance zone. However, through experimentation, it was determined that as the robot made a right-hand turn following a wall, sometimes the left front sonar would detect a wall and force the robot to initiate a left hand turn prematurely, possibly causing the robot to skip an entire section of the maze. To reduce the possibility of this occurring, the distance for this sonar was further reduced to just 9 cm. With this reduction, the robot was capable on entering all corridors and all rooms in the Trinity maze.

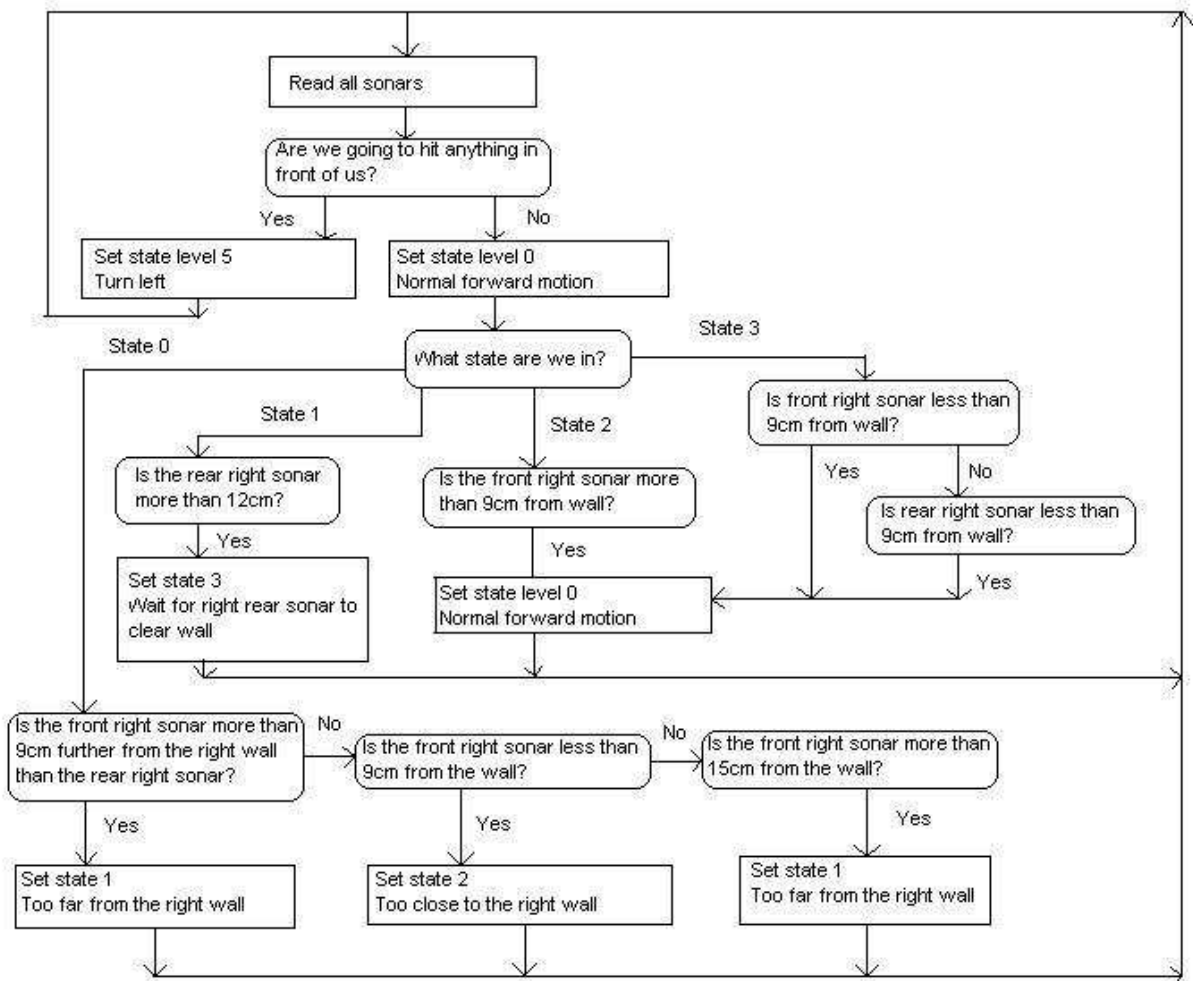


Figure 4: State Diagram

An important design consideration made when developing how the robot should turn was the relative speed of each of the motors powering the robot's tracks. A need for a left-hand turn implied an imminent collision with some object in front of it. So it was necessary to not move forward any more until we had a definite and clear path to move forward. Therefore, the front collision avoidance check (state 5) was always done first prior to any other state transitions being allowed.

However, during right-hand turns, this turning method did not suffice. The robot would consistently turn too quickly or too slowly, causing it to collide with a wall and in turn damaging the attached sonar assemblies. Therefore, it was decided the robot would alter the power ratios driving the left side motor at full speed, while reversing the direction of the right motor and decreasing its power level to 2. In doing so, the robot still moved forward just enough, while still turning right. This "arcing" maneuver allowed the robot to follow the right-hand wall through a 90-degree turn. It also was capable of handling the difficult 180-degree turn noted above. Therefore, if no walls are detected initially, the robot will slowly arc out in a gradual right-hand turn and wait until one of its 5 sonars detects an object, at which point the normal navigation state machine will be used.

Physical Robot Structure

The physical layout of the robot came into being after several iterations of the robot chassis. From a 2-wheeled device to the final treaded version, several physical implementation issues were discovered and surmounted to attain a stable and expandable platform for future robot development. We based our robot chassis on LEGO Technic materials thanks to the ready availability of LEGO motors and the easy reconfiguration of models built from LEGO elements.

Initially a 2-wheeled platform was built with a caster wheel in front to balance the entire system. The idea was to vary power to either side of the robot chassis to cause the robot to turn in a specific direction. While this is feasible, we found that the caster wheel, built out of LEGO parts, was not strong enough to withstand the torque put on it when the chassis turned. We attempted to remove the front caster wheel with just a peg with a smooth surface on the bottom, but it too would break off quite easily during turns. The system also had a very high center of gravity, allowing any failure of the wheels' axle or front wheel/peg systems to cause significant damage to the circuitry riding on top of the platform.

We redesigned the platform to be much lower to the ground and to use a treaded maneuvering system. We no longer had to worry about balancing the device with a caster wheel or peg, but still retained the advantage of having just 2 motors power the entire platform. It also reduced the height of the system dramatically, allowing any LEGO-based failures of parts to be minimized on the rest of the system. We also now had a larger workspace on which to add additional hardware components.

One of the first things we noticed in testing the robot chassis was that the power sent from the LEGO motor to turn the wheel inside the tread was not that great. It was decided that a gearing system would be added to the motor mechanics to increase the amount of torque sent to the tread wheels. Several different gearing sizes and designs

were tested and finally came to the decision that just one extra gear, a 8-toothed gear wheel, would be sufficient to attain a sufficient torque level.

When we had our prototype tank tread robot complete we discovered that we were under powering the drive motors. To compensate, a modification was made to allow the Cricket to run off a 9v source. Wiring the Cricket's battery pack to our bus device's battery pack created this higher voltage source. While this did fix our sluggish motors, it caused some power problems on the bus device. The Logo Chip would reboot whenever the motors were turned on. To solve this problem a power regulator was added to the bus device as well as several capacitors. While debugging the sonar device hang related to the Cricket Bus, we also placed a resistor in line with the bus data wire. While this did not fix what ultimately turned out to be the bus timing conflict, it does help to further electrically isolate the Logo Chip from the Handy Cricket.

With the power systems of the robot set, we encountered an additional problem. After the robot was operated continuously for approximately 30 seconds, the axle systems would begin to separate due to the torque force put on them during turning movements. While this event was somewhat expected due to the characteristics of a LEGO-based device, the robot would constantly have to be stopped, tightened, and restarted. This proved to be a consistent hassle during development so the offending parts were glued together with a very small application of adhesive. After curing, the robot handled much better and the gearing and axle systems were never altered again.

Sonar placement on to the robot chassis was initially fairly natural and obvious. As stated earlier, 3 sonars were used in the front of the robot and 2 on the side. However, when mounted low to the ground, the sonars, especially the front sonars, would get erratic sensor readings. After some inspection, we found that the sonar cone was detecting the floor as an obstacle and causing incorrect navigational errors. The solution was simple: raise the sonars high enough on the platform so they would not misinterpret the floor as an obstacle.

The angle of the front sonars is also very important. As stated earlier, 2 extra sonars mounted at 45-degree angles from the main front sonar were added to avoid some cases where the robot sensors would have a gap in detection. However, due to the nature of the LEGO system, parts are not easily placed at 45-degree angles. We did manage to rig a system together where we got an approximate 45-degree angle, but we relied on a few pieces of electrical tape to ensure stable placement. Occasionally, the sonar configuration moves just enough to allow the sonar cones to have a gap in sensor readings and the robot can potentially hit a wall during a turn. This sonar layout system would need to be improved in later revisions of the chassis.

When the robot made arcing right turns initially, we found that it would make turns too wide and not track close enough to the right-hand wall. This impacted maze navigation performance because if the robot didn't turn tight enough, the front sensors had the possibility of detecting the left-side wall during the turn, causing the robot to spin left and exit the area of maze altogether. Through experimentation, we found that by moving the second side sonar closer to the front of the robot chassis, the second sonar would detect a clear area and enter state 3 and begin to turn right earlier. Originally the 2 side sonars were positioned at either edge of the robot, but by the end of this testing, resided almost directly next to one another in the front quarter of the chassis.

Some experimentation was done to see if having an additional motor on either side working in tandem increased speed, performance, or traction. We did not see a significant improvement in robot operations, so we removed the additional motors for the time being to conserve battery life during operations. In the future, extra motors may be added if necessary to counter additional weight added to the robot or to increase the amount of torque applied to each tread.

Conclusion

Upon the completion of our time allotted for work on the project, we had reassessed project milestones and completion dates and overcame several issues that we did not expect to encounter at all. In the end, the robot satisfied our project requirements and is readily available for further enhancements. With the use of just 5 sonars, we demonstrated basic maze navigation in less than 1k of compiled Logo code.

Acknowledgments

The components for this project were provided by the University of Massachusetts, as part of the Robotics I course (91.548). We would especially like to thank Dr. Fred Martin for his expertise in the lab, as well as his dedication to the students in the class.

References

¹Mike Linnen, "Building A Maze Robot,,: <http://www.parex.org/buildmazerobot.shtml>.

²The Handy Cricket Home Page: <http://handyboard.com/cricket>

³LogoChip: <http://www.wellesley.edu/Physics/Rberg/logochip>

⁴The Devantech SRF04 Ultrasonic Range Finder Timing Diagram: http://www.robotstorehk.com/srf04_timing.pdf