Developing an Explorer Bot with RCX to PC Communication
CPSC 643 Robotics Programming Term Project
Jack Perdue August 6th, 1999

1 Overview

For this project, I studied communications between the Lego Mindstorms Robotics Invention System (RIS) RCX unit and a host computer. The objective was to determine how to overcome the limitations of the RCX in terms of variable types, data structures, memory capacity and processing power. I hoped to demonstrate this by developing a a Lego "bot" that would explore a room, report back to a controlling PC what it had found, and either continue with its present programming or receive alternative programming by the PC. By maintaining a constant conversation with the PC, I managed to make it possible for the bot to retrieve some data about the environment and report it back to the PC. However, the ability for the controlling PC to digest the environment information and provide a new program was not implemented due to time constraints and the inaccuracy of the data the bot was reporting.

2 Methodology

Given the limited availability of the Mindstorms kits, I set out the first weekend of class to learn as much as I could about what I might be able to build.

Russell Nelson's *Lego Mindstorms Internals* page [RNLMI] provides an abundance of information that couldn't be found at Lego's Mindstorms web site. [LGLM] Another extremely helpful resource are the newsgroups available at lugnet.com [LUGNET]. That's where you will find most of the people listed in the bibliography of this project discussing future directions in Lego RCX programming. It is also a great place to go for information, as I found when I wanted to overcome the problem with the IR tower falling asleep (which is covered below).

Once I had an overview of what the RCX was capable of, I then set down to try some basic communication between the RCX and PC. I originally planned to make a page scanner that could scan a page of arbitrary (very large) size. The idea was to have the robot use the light sensor to do the scan and then report back what it found to the PC. I had seen at Lego's web site a project that someone had done that could scan something the size of a sheet of paper. I wanted to develop a bot that would be put on a very large piece of paper, have it find a corner and align itself to the edge, and then proceed to scan line by line while sending the results back to the PC. However, I abandoned this idea when I realized a single Mindstorms kit didn't provide all that I needed to do it right (I needed more light sensors at least).

So, I then changed the plan to a basic room explorer bot. Have the bot walk around the room bumping into things and making course corrections. Additionally, have it report back to the PC the contacts it had made so the PC could provide better instructions for those parts of the room that needed more exploration.

Having not searched the lugnet.com discussion groups at that point, I assumed the range of the IR tower was rather limited. Therefore, I planned to build an explorer bot that could carry my Compaq laptop around so that it would always be in range of the PC.

So, with that in mind, I sat down with just the RCX, a couple of motors, a couple of touch sensors, my laptop and the IR tower to figure out how to get a dialog going.

I originally considered three approaches:

- Kekoa Proudfoot's RCX Tools [KPRT]
- the communication options of Dave Baum's Not Quite C (NQC) [DBN]
- Scott Lewis's RCXPort [SLRP]

Kekoa's tools include a simple program named send.c which provides for basic communication with the RCX. Dave's NQC provides an easy way to send messages and raw commands (much like Kekoa's). Scott's RCXPort is a Java interface to the RCX.

Given the time constraints imposed, I quickly discarded Scott's RCXPort since he had only tested it with JDK 1.2 and I was only familiar with JDK 1.1 (and that familiarity has waned in the year since I last used it). Kekoa and Dave's tools were available as C source which I am much more comfortable with when under deadline.

I experimented with the two remaining approaches. At first, I thought I'd try using the RCX's "message" routines. This had a couple of problems. First off, there were timing/coordination issues that would have to be resolved in order to keep the RCX and PC from talking at the same time (and taking corrective action when they realized that they had). However, the second problem of limited communication bandwidth made me abandon this approach altogether. I found that Dave's NQC took about 350ms per message while Kekoa's took about 450ms. This meant that only two to three messages per second could be sent. I managed to get Kekoa's send() to go a little faster by lowering some of the timeout values, but I know that he probably picked the values that he did for good reason and I was possibly affecting the reliability of his code by modifying it. I also noted that most of the time spent in Kekoa's code was waiting for the last byte of a message that never came. I thought it might be easy figure out how many bytes to expect on the reply, and thus forego the final byte, but, again, making such a change would have affected the robustness of his code. In the end, since it was unknown at what rate the RCX would generate data, and since there were the coordination issues to overcome, I decided to toss this approach.

I then focused on using the RCX "get value" command in conjunction with the RCX's Datalog to provide polling and data exchange between the PC and

RCX. This had two advantages and one major disadvantage. The first good point was that coordination was no longer an issue since the PC was responsible for initiating all communications. The other benefit was the communication "burst" used to transfer the Datalog that allowed for more effective use of the bandwidth available. The downside to this approach is that it only works between the PC and one RCX. A multi-RCX implementation can not use this method since all RCX's would attempt to respond to the PC's polls. In the future, I think that the Lego Network Protocol being discussed in the newsgroup lugnet.robotics.rcx.legos for incorporation into the LegOS (Lego Operating System) might be the best way to go. However, they are still working out the details on how to implement it (and, of course, looking for volunteers to contribute code).

Although Kekoa's send.c seems to be a bit more powerful in terms of message handling, Dave's NQC provided all the functionality I needed. Since I had already decided to use NQC to write the RCX code, I decided to go with a "single-vendor" solution and use NQC exclusively.

I should mention the development environment at this point. I generally run a Win95 desktop with the laptop on the local net running RedHat 5.2 Linux. I used Mark Overmars RCX Command Center[MORCC] on the Win95 side to develop the RCX code (in particular, because he has a great tutorial[MONT] on NQC). Once it compiled clean, I would FTP it over to the laptop where it would be compiled again (using the Linux NQC) and then downloaded to the bot. In theory, the code developed should work under Win95 or Linux. The "testing grounds" was the kitchen floor which is adjacent to my study. With the IR tower facing out the doorway of my study, I was able to communicate with the RCX in most parts of the kitchen.

I should also note that I was easily able to port NQC to HP-UX (Hewlett-Packard's Unix). The actual communications didn't work (I didn't see any point in trying to make them work), but it does provide a way to at least check the NQC code for compile-time errors. I imagine it would be very simple to port NQC to other Unix variants (e.g., Solaris).

Once I had some preliminary success with polling the RCX and retrieving the Datalog, I then investigated how to build the laptop carrying bot.

After several failed attempts, I finally managed to develop a bot that was big enough and had enough torque to carry a book [CLRIA] that was the same size and about half the weight of the laptop. However, the gear ratio required to move the weight made for very slow progress across a room. Since I wanted to be able to demonstrate the project in a reasonable period of time, and since I did not think I could build a sturdy enough platform that I would trust to hold a $2500 laptop as it roamed around the room ramming into things, I abandoned the laptop explorer bot.

It was at this point that I went to the RCX group at lugnet.com to research the range of the IR tower. I found out there that I could reasonably expect a 20 foot range in a room like HRBB 104, and so decided to change my tack and develop the bot I ended up with.

So, again I ripped apart the bot I had (for about the sixth time at this point [the calluses that I developed on my fingertips have just peeled off this last week... my fingers were very tender at this point in the development due to prying apart bricks]).

I concentrated this time on making a rock-solid drivetrain. Previous designs had suffered over time due to loosening of bricks and the different forces placed on different axles. After some experimenting, I came up with a very economical package that clamped the motors, gear and wheel axles very tightly so that their wouldn't be room for the bricks to move. With the drivetrain I developed, I was able to run for extended periods of time without gears slipping (in fact, they never did in the tests I did... only time would tell if they would eventually work themselves loose).

I then worked on a feeler array that would be responsive to obstacles encountered. My first approach to this was a dismal failure, but the lessons learned in the experience allowed me to improve upon my design.

I experimented with different free-wheels at the front of the bot for support. After failing to find one I found satisfactory, and noticing in the Mindstorms Constructopedia that the round, smooth, black thing was designed to be a skid plate, I decided to take that approach. Since the demo was to take place on a well-waxed Texas A&M University classroom floor, I figured the coefficient of friction would be sufficiently low for this to work.

Once I had the bot, it was time to go back to developing the software.

I needed a way to keep the IR tower "awake". I had read in the lugnet.com discussion groups that it had a tendency to go to sleep when idle in order to conserve batteries. Furthermore, the RCX could do nothing about the situation, only the PC could wake up the tower. That lead to the decision to poll the RCX constantly in order to keep the tower awake. Since the polling was taking place anyway, I decided to put it to some use.

The first, rather minor, use I found for the RCX's responses to the polls was the use of a proximity detector. I had read that the light sensor could catch the reflection of IR messages off of objects and that this could be used to build a proximity detector of varying ranges. When I had first played with this, I had set up a task in the RCX to constantly send out a message from its IR port and then check the light sensor for the reflection. However, this had the disadvantage that when the PC needed to talk to the RCX, the RCX never heard it because RCX's constant sending would collide with the incoming messages from the PC. However, now I had the RCX providing a constant message stream in cooperation with the PC. Since this was rather ideal, I decided to slap a light sensor on the RCX facing upwards to detect any overhead items (like a desk or chair).

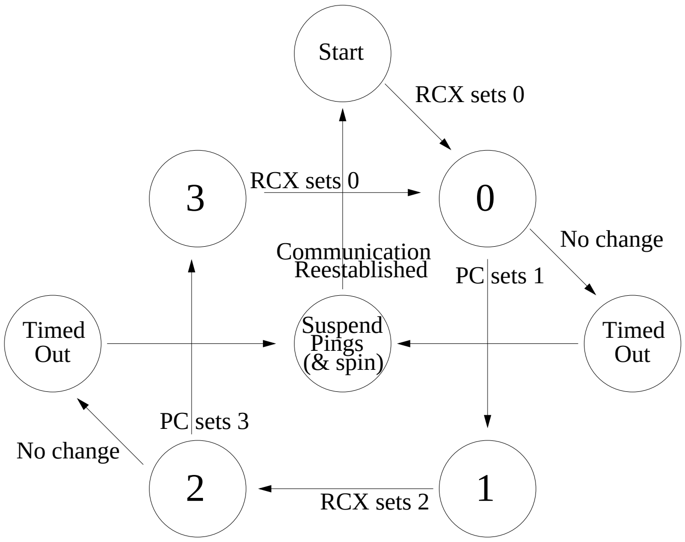

Figure 1: Planned PC/RCX Ping Routine State Diagram

The second, more important, use I found for the RCX polls was to keep the RCX informed as to its range to the PC. I wanted the bot to maintain a constant dialog with the PC so that it could notify the PC when its Datalog was full and needed to be collected. If contact was lost, the bot would have simply stopped (as programmed) because the Datalog was full but nothing was retrieving it. To this end, I came up with a simple little protocol which has the PC and RCX constantly updating variable 0 in the RCX to let each other know what's going on. Originally, I was just going to use variable 0 to notify the PC that the Datalog of recorded observations was available. However, this new ping protocol also allowed for the bot to quickly determine when it had just gone out of range so that it could spin around until facing the RCX (and hopefully, regaining communication) before continuing in its explorations. Figure 1 provides a state diagram of the protocol developed.

3 Deficiencies and Future Directions

There are a number of problems with the current design that make the data it collects almost worthless in terms of getting an accurate picture of the explored environment. There remains a number of improvements that can be made to the hardware and software to enhance the bot's exploring.

3.1 Drivetrain

The first issue that needs to be addressed is straight tracking. The present two motor to two wheel design suffers because the two motors do not produce identical speed/torque. Applying full power to both motors should make the bot go straight if both motors are precisely calibrated. However, that's not the case.

One way to overcome this problem is with the use of an "adder-subtractor" differential like the one designed by Leo Dorst[LDAS]. Unlike the standard differential that directs power to the wheel experiencing the least resistance, the adder-subtractor provides a way to precisely control the left and right wheel axle when straight forward tracking is desired. The standard differential is designed for bots that might use a drivetrain similar to an automobile: two-rear wheels providing forward velocity and the two front wheels providing steering. The standard differential will direct the most power to the wheel experiencing the least resistance. This is good for the automobile (unless it gets stuck in the snow/sand) as it directs the power to the outer rear wheel on a curve. However, that is not what is wanted for the explorer bot.

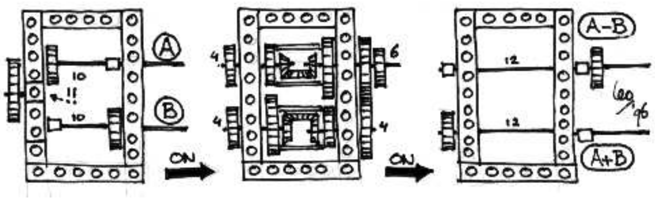

Figure 2: Leo Dorst's Adder-Subtractor[LDAS]

The adder-subtractor differential uses two of the differential gears (the Mindstorms kit only has one) and two motors to provide two axles of output, "A+B" and "A-B". In order to go straight, full power would be applied to motor A while motor B was held stationary ("frozen"). Then the two wheels receive the total output of motor A. To initiate a turn, motor B is either run in forward or reverse. For example, if the A+B output is hooked to the left wheel, the A-B output is hooked to the right wheel, motor A is run at full and motor B is run at full, then the output of A+B axle is approximately twice the output of just the A motor alone, the output of the A-B axles is approximately zero and the robot will turn to the right.

Another improvement that could be made in the current design is lowering the gear ratio from the motor to the wheels. I originally was investigating how to build a bot that would be able to carry a fifteen pound laptop around with it while it explored (in order to keep the RCX in range), so I originally required a lot of torque to move this mass. When I made the transition to a non-PC-carrying bot I did make adjustments to my gearing to get more speed, but I believe the present design still has too much torque. I base this belief on the fact that at present the 7 motor speeds that NQC provides all seem to be the same. That is, the present transmission provides so much torque (compared to the weight of the bot) that the motor is not burdened at low speed and therefore the wheels run the same speed whether the motor is set for low or high power.

This meant that turns had to be implemented by either stopping or reversing one of the motors. Attempting to turn by running one motor at forward full and the other running forward at low-speed resulted in the bot going straight instead of turning. It would be nice if this was corrected so that turns while traveling forward could be implemented.

3.2 Feelers and Touch Sensors

The second issue that can easily addressed is the addition of a touch sensors on the sides and rear of the bot in order to detect collisions when backing up or turning. The present design suffers in that it does not detect these collisions causing inaccuracies in the reported distance traveled. If it could detect these rear and sideways collisions, it could stop the motors immediately (after logging the contact) and take corrective actions.

The quickest sensor addition is simply putting a single touch-sensor/feeler combination on the rear of the bot. This would take a third touch sensor (the Mindstorms kit only has two) and could be connected to the RCX where the light sensor (for overhead radar) is currently connected.

A far better solution would be use one of the tricks another Lego enthusiast has found. By hooking three touch sensors to a single RCX input via three different resistors, and then monitoring the sensor level, there is a way to determine which of the three sensors was activated. By utilizing two of these three sensor combinations (on two RCX inputs), a bot could be built with 2 front, 2 side and 2 rear touch sensors, which would be a vast improvement over the present design. Furthermore, it would leave the third RCX input available (e.g. for a rotation sensor).

This last mentioned solution touches upon a method of sensor monitoring that should probably be implemented anyway. The easiest way to program the RCX to read touch sensors is in terms of a default threshold which allows for simple On/Off detection. A more sophisticated approach would be to monitor the level of the output from the sensors directly and then act accordingly as the value changes. By having more control over the threshold, the variability in sensor sensitivities might be accounted for in software instead of hardware (I had to tape a piece of paperboard to one of the feelers because I found that one of the sensors was significantly less sensitive than the other one).

One final improvement is the mounting of the feelers on independent axis. The current design has both the left and right feelers swinging about a common axle. This has a problem when the bot hits a flat surface head on as the force on both feelers fails to allow for rotation on the single axis, thus preventing the touch sensor from registering the obstacle. The easiest thing to do for the present design is to provide another axis up front and separate the feelers.

3.3 Motion Detection

Although the addition of touch sensors might improve upon the bots ability to detect obstacles when moving, in order to further improve upon the data collected, it would be useful to verify that the bot is actually moving when it thinks it is (i.e., when its motors are running).

The quick fix for this would be the addition of a free wheel on the front (instead of a skid plate), attached to a rotation sensor. Then it could be monitored to ensure that when the motors activated, motion is actually taking place (i.e., the bot isn't hung up on an obstacle). Some quick changes to the RCX explorer code could have the bot stop immediately when it detects that it's activated motors are not generating motion on the free wheel. This would be very helpful in low traction situations where the bot doesn't have enough grip on the surface to generate the force necessary to activate the touch sensors when pressing against an obstacle.

Again, this isn't possible with just the Mindstorms kit, as it does not include a rotation sensor.

3.4 Location Detection

The bot does have a basic location detection scheme. When it gets out of range of the IR tower, it spins until it is facing it again and then heads that way. It's not very accurate, but at least you know that the bot is somewhere in the range of the IR tower (which, if you know the location of the IR tower, gives commercial grade GPS resolution).

In order to get a more accurate picture of the explored environment, some type of location detector would be an improvement. Obviously, military grade GPS isn't in the budget. Perhaps something could be done with a light source (such as IR) to provide a way for a robot to triangulate its location within a fixed space. However, this would probably require more math than the RCX is capable on its own (though the PC should be able to help). I haven't really investigated the best approach to this given a Mindstorms kit.

3.5 Communication and Ranging

Although I have a good idea of how I wanted the bot to behave in in terms of keeping in contact with the IR tower, I did not fully implement it. There are some parallel task and logging issues to resolve, but it shouldn't be hard to implement the spin until communication restored routine. It's a bit difficult because it will require notification to the motor driving routine that the pings have been lost and that the current path needs to be abandoned (and logged) before spinning (and logging the spin) to restore communication. I've hesitated to implement the changes that I think would accomplish this since I didn't have an RCX available to test the modified code (and therefore, the code has not been changed).

When the Lego Network Protocol has a bit more substance to it, I think it would be nice to incorporate it into this project. The present design is limited to one bot and thus the communication range of one bot. It would be nice to have "courier" or "relay" bots that ferry messages from distant (out of range bots) back to the PC. This change, however, would take a significant overhaul of the present bot code to facilitate the forwarding of messages from RCX to PC and vice-versa.

4 Additional Information

There are pictures, comments, links and source code available at my web page documenting the bot[JKPEB].

References

[DBN] Dave Baum. NQC Home Page. http://www.enteract.com/~dbaum/lego/nqc/index.html.

[CLRIA] T. H. Cormen, C. E. Leiserson and R. L. Rivest. Introduction to algorithms. MIT Press and McGraw-Hill Book Company. 1992.

[LDAS] Leo Dorst. (Lego) Adder-Subtractor. http://carol.wins.uva.nl/~leo/lego/diff.html

[LGLM] The Lego Group. Lego Mindstorms Home Page. http://www.legomindstorms.com/

[LUGNET] The Lego User Group Network. http://www.lugnet.com/

[SLRP] Scott Lewis. RCXPort - Java Interface to the Lego Mindstorms RCX. http://www.slewis.com/rcxport/

[RNLMI] Russell Nelson. Lego Mindstorms Internals. http://www.crynwr.com/lego-robotics/.

[MORCC] Mark Overmars. RCX Commnad Center Home Page. http://www.cs.uu.nl/~markov/lego/rcxcc/index.html

[MONT] Mark Overmars. NQC Tutorial. http://www.cs.uu.nl/~markov/lego/.

[JKPEB] Jack Perdue. Jack's Explorer Bot. http://www.cs.tamu.edu/people/jkp2866/cpsc643/explorerbot.html

[KPRI] Kekoa Proudfoot. RCX Internals. http://graphics.stanford.edu/~kekoa/rcx/.

[KPRT] Kekoa Proudfoot. RCX Tools. http://graphics.stanford.edu/~kekoa/rcx/tools.html