LMHanoi

Tower of Hanoi with Lego Mindstorms

Markus Flatz, Peter Palfrader, Andreas Rottmann
forename.surname@stud.sbg.ac.at

Report
created during the course
Embedded Software Engineering
winter semester 2010/11
Lecturer: Univ.-Prof. Dipl.-Inform. Dr.-Ing. Christoph Kirsch

University of Salzburg, Master program Applied Informatics

Contents

1 Introduction **2**

1.1 The puzzle 2

1.2 Lego Mindstorms 2

1.3 NXC 2

2 The robot **3**

3 Algorithms **4**

3.1 Recursive algorithm 4

3.2 Iterative algorithm 4

4 Software **5**

4.1 Application logic 5

4.2 Inter-thread communication 6

4.3 Robot controller 6

4.4 Music 6

5 Experiment **7**

6 Closing remarks **8**

A Subproject responsibilities **8**

Abstract

The project goal was building a Lego Mindstorms robot capable of solving the puzzle "Tower of Hanoi". Our apparatus has a fork to pick up the disks, it can be moved in three directions and relies on color and touch sensors to determine its position. Using the NXC language, we created a three-threaded control software with one thread for application logic, one for the actual robot control, and one for situational background music. The moves are determined by an iterative algorithm yielding each movement decision in constant time.

1 Introduction

1.1 The puzzle

The Tower of Hanoi is a stack of different-sized disks. There are three rods. In the beginning, the first rod contains all the disks. The largest disk lies on the bottom, the second largest one is directly above and so on, the smallest disk lies on top. The goal is to move the whole tower to the third rod. In each step, only the topmost disk of one rod can be moved onto the top of another rod's stack. Additionally, it is never allowed to pile a disk onto a smaller one.

1.2 Lego Mindstorms

Lego Mindstorms[1] is a kit to build embedded systems consisting of a programmable brick, motors, sensors and a number of ordinary Lego parts.

The current version, Lego Mindstorms NXT 2.0, relies on a 32-bit ARM7 microprocessor, 64 kB RAM and 256 kB Flash memory and offers three output ports and four input ports, sound and connectivity via USB and Bluetooth. Three servo motors with rotational sensors, two touch sensors, one color sensor and one ultrasonic sensor are included in the box.

A graphical programming environment based on LabVIEW ships with the kit.

1.3 NXC

NXC[2] (Not eXactly C) is a programming language similar to C to write code for the Lego NXT brick. It relies on the assembly language NBC (Next Byte Codes). The compiler is released under the Mozilla Public License.

We used this language to solve our problem, since it is more versatile and allows more control than the graphical tool included in the kit.

[1] `http://mindstorms.lego.com/`
[2] `http://bricxcc.sourceforge.net/nbc/`

2 The robot

Our robot is built of parts of the Lego Mindstorms NXT 2.0 set 8547 and additional Lego bricks, except for the puzzle itself, which is made of wood. The construct is pictured in Figure 1.

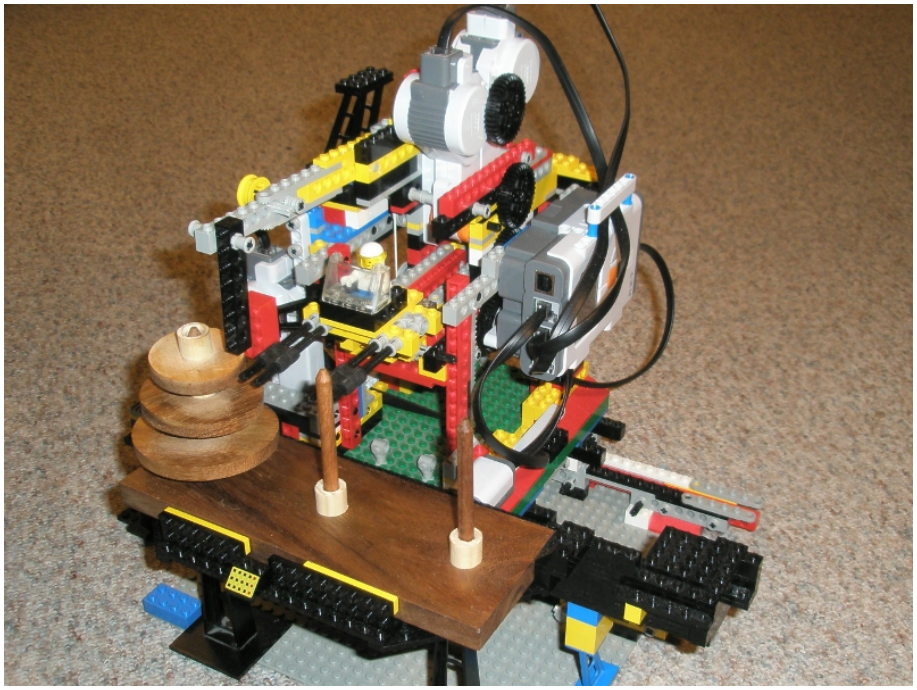

Figure 1: The robot, Klaus

Spacers between the disks allow the robot to pick them up with its fork, which can be moved along three axes by motors.

One motor shifts the whole wagon (the construction on the green plate in Figure 1) left and right between the three pegs. It relies on a color sensor to recognize colored Lego bricks marking the three pegs and an additional outer position used in the initialization and termination phase.

The second motor moves the shuttle (the inner part, including the fork) up and down along toothed racks. A touch sensor is triggered at the uppermost position and also when an obstacle, i.e., a disk, is encountered while moving downwards.

Finally, a third motor is responsible for forward and backward motion of the fork. We also use the ability of the NXT brick to play sounds, which is described in section 4.4.

3 Algorithms

3.1 Recursive algorithm

The Tower of Hanoi problem is a common example used to introduce computer science students to the concept of recursion, and as such the recursive solution to the problem is well known. A possible formulation of the algorithm is shown in Listing 1, where n, the first argument, is the number of disks.

```
1 void hanoi(int n, rod source, rod help, rod destination) {
2     if (n > 0) {
3       hanoi(n - 1, source, destination, help);
4       move_disk(source, destination);
5       hanoi(n - 1, help, source, destination);
6     }
7 }
```

Listing 1: Recursive algorithm

Theorem 1 (Number of moves)**.** *Let n be the number of disks. Using the recursive algorithm from Listing 1, the number of moves is $2^n - 1$.*

Proof by induction. For $n = 0$, the number of moves is $0 = 2^0 - 1$.
For $n > 0$, the number of moves is $(2^{n-1} - 1) + 1 + (2^{n-1} - 1) = 2^n - 1$. □

Moreover, this movement sequence is optimal, i.e., the shortest possible.

3.2 Iterative algorithm

One way to solve the problem without recursion is presented by Buneman and Levy[2],[3]: The three rods are imagined to lie on a circle. If the number of disks is even, the rods source, help and destination are ordered clockwise, if the number is odd, they are ordered counterclockwise. Listing 2 depicts the algorithm in pseudocode.

```
1 while (not all disks on destination rod) {
2     move the smallest disk one rod clockwise;
3     if (a disk other than the smallest one can be moved) {
4         move this disk;
5     }
6 }
```

Listing 2: Pseudocode for the iterative algorithm

This algorithm yields the same, optimal movement sequence as the recursive algorithm from Listing 1. All steps are well-defined and deterministic. Especially the condition in line 3 always holds for exactly one disk during the run, and for no disk if all disks are on the destination rod, i.e., when the puzzle is finished.

The move in line 4 is clearly defined, too, since the disk cannot be piled onto the smallest disk.

As the Tower of Hanoi is a very popular puzzle, many more algorithms to solve the problem exist.

4 Software

Our application runs in three threads, called *tasks* in NXC lingo. Figure 2 shows these threads and how they communicate.

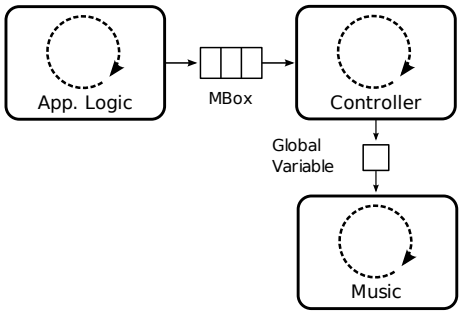

Figure 2: General structure of our program

One thread is responsible for the "application logic", i.e., it makes decisions on disk movement. The second thread, called robot controller, is responsible for implementing these decisions, responding to sensor inputs and issuing motor commands. The last one is challenged with providing entertaining and informative music during the run of the program.

4.1 Application logic

In the beginning, we tried to apply the well-known recursive algorithm described in section 3.1. Unfortunately, it turned out that the NXC language does not support recursion. Fortunately, the iterative solution from section 3.2 can be implemented without any problems.

Our first, naïve implementation used a single array with n elements, where n is the number of disks, storing the current location of each disk. The task of finding a movable disk required scanning this array, resulting in a time complexity of $O(n)$ for a single movement decision.

Our robot cannot support large numbers of disks because of its limited size. Nevertheless, we wanted to refine the implementation to yield each disk movement in constant time. We achieved this goal by using more memory, in a classical space-time tradeoff. Although the memory consumption is now approximately three times as high, it is still linear in the number of disks, so we considered this expense justified.

The basic idea is using an array of size $r \cdot n$, containing the current disks (at most n) on each of the r rods. Additionally, an array with r elements holds the current height of each rod's stack, allowing direct access to their topmost disks. It is only necessary to regard these disks to find the next move, therefore the time complexity per move is $O(r)$, i.e., $O(1)$, since r is constant ($r = 3$).

4.2 Inter-thread communication

While the application logic thread runs without any prerequisites, the calculated disk moves must be announced to the robot controller. For this sake, we created a mailbox containing movement instructions. Such an instruction specifies a source and a destination rod. The size of the disk is not enclosed, since the robot controller can only ever move the topmost disk anyway. No further information is needed.

The mailbox provides a put and a get function, and is internally implemented as a circular FIFO buffer of configurable size. To guarantee correct behavior in a multi-threaded context, two counting semaphores (for states empty and full) are used. NXC provides only binary semaphores (mutexes), so we had to create a counting semaphore ourselves. Trono et al. provide an interesting insight into the difficulties of this task in [1]; we implemented the method proposed by Barz, shown there in Figure 2.

Besides this, the music thread also depends on status information from the robot controller. However, this need can be simply satisfied with a global variable.

4.3 Robot controller

The robot controller processes moving instructions one by one. A disk move always follows the following pattern: moving the wagon to the source rod, lifting the topmost disk, moving to the destination rod and dropping the disk. These steps are further refined to functions containing actual sensor checks and motor commands. The installed components are described in section 2.

4.4 Music

Besides the core task of solving the puzzle, a third thread uses the ability of the NXC brick to issue background sounds. We did this for entertainment, as well as for the challenge of additional concurrency.

As briefly mentioned in section 4.2, the music is situational, depending on the current state of the robot. There are different themes for initialization (no sound), wagon movement, wagon halt or vertical movement, finishing a disk move and completion of the puzzle.

The music thread reads a global variable set by the robot controller, then plays the corresponding theme. The variable is read again when the theme is finished, a theme is never interrupted.

5 Experiment

Beyond the ability to actually solve the puzzle, we tested whether our robot shows the expected timing behavior in dependence of the number of disks. Theorem 1 states that the number of moves is $2^n - 1$, where n is the number of disks. Our experiment shows that this property holds true for our robot as well, suggesting that we implemented the algorithm correctly.

We were interested in the actual execution times for $n = 1, 2, 3, 4$ disks. For this we simply set a constant in the code and provided the appropriate number of physical disks. The mechanical dimensions of our robot allow only up to three disks, so the experiment with four disks is partially simulated by manually triggering the touch sensor when the robot wants to move the lowermost disk. The results are shown in Figure 3. The red crosses labeled with *'lmhanoi.dat'* are the measured values, while the green line $expreg(x)$ is a calculated regression function.

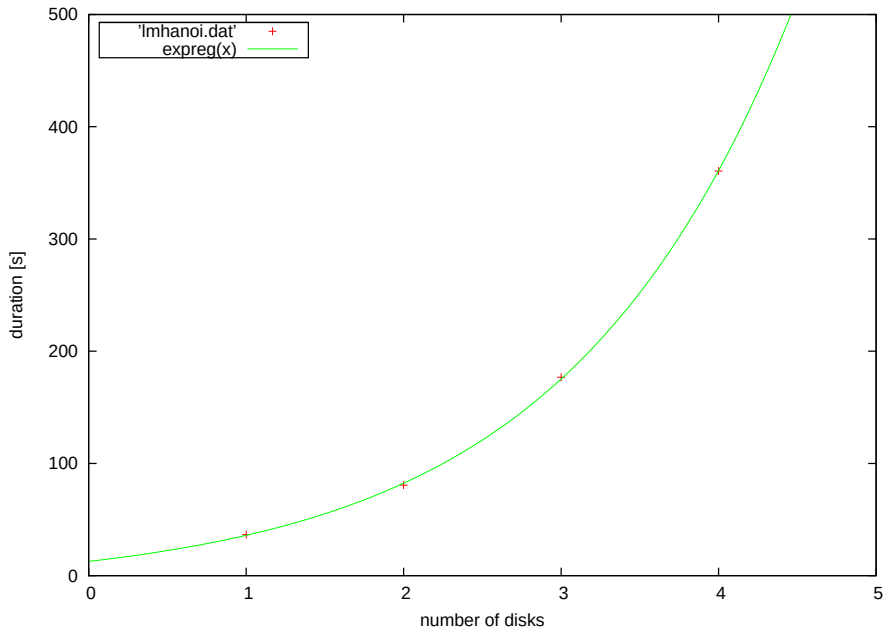

Figure 3: Puzzle-solving duration for different numbers of disks

The expected behavior is $duration(n) = (2^n - 1) \cdot d + c = (e^{n \cdot ln(2)} - 1) \cdot d + c$, where d is the mean time per disk move and c is a constant initial and terminal overhead. This leads to the regression function

$$expreg(x) = (e^{0.693 \cdot x} - 1) \cdot 23.2 + 12.7$$

which approximates the data points fairly well. (Final sum of squares of residuals: 6.11.)

6 Closing remarks

The presented robot is able to successfully solve the Tower of Hanoi puzzle. Our project page[3] provides the NXC source code, as well as a short demonstration video.

A Subproject responsibilities

Most of the work on the project was done during regular team meetings, where we sat together discussing and solving the problems at hand. So, all team members contributed to nearly all parts of the project. Nevertheless, we tried to create a list giving the fields each person made the most important contributions to.

Markus Flatz:

- algorithms and application logic
- music and music thread
- experiment
- report and project page

Peter Palfrader:

- robot construction
- robot control
- counting semaphores
- application logic

Andreas Rottmann:

- mechanical issues
- concurrency
- inter-thread communication
- drawings

[3] `http://www.cs.uni-salzburg.at/~ck/wiki/index.php?n=ESE-Winter-2010.LMHanoi`

References

[1] John A. Trono and William E. Taylor. Further comments on "a correct and unrestrictive implementation of general semaphores". *SIGOPS Oper. Syst. Rev.*, 34:5–10, July 2000.

[2] Wikipedia. Tower of Hanoi — Wikipedia, The Free Encyclopedia, 2011. `https://secure.wikimedia.org/wikipedia/en/w/index.php?title=Tower_of_Hanoi&oldid=407310252`.

[3] Wikipedia. Türme von Hanoi — Wikipedia, Die freie Enzyklopädie, 2011. `https://secure.wikimedia.org/wikipedia/de/w/index.php?title=T%C3%BCrme_von_Hanoi&oldid=83701385`.