

LOGO-like 'Turtle' control with iRobot Create and ZNEO

Michael Shick



May 6, 2011

Abstract

LOGO is an educational programming language known for its Turtle Graphics system for imperative graphics drawing. Programs in LOGO for the Turtle contain statements such as `FORWARD 100` or `LEFT 90`, which the Turtle then interprets and follows, leaving a trail behind it. Students could use the trail and computer science concepts to create artwork and interesting patterns. Although drawing was generally done virtually with a virtual Turtle, physical Turtle robots were also made to illustrate the concept outside of the virtual world. This project implements such a Turtle robot with iRobot's Create robotics platform and the ZNEO contest board.

Contents

1	Status	1
2	Design Overview	2
2.1	LOGO	2
3	Hardware	3
3.1	Materials	3
3.2	Setup	3
4	Software	4
4.1	Drivers	4
4.2	LOGO Implementation	5
4.3	iRobot Create Control	6
5	Retrospective	7
5.1	Improvements	7
5.2	Lessons	8
6	Gallery	9

1 Status

The project is incomplete; but the most important components are all present. The basic LOGO functionality works, allowing arbitrary sequences of movement to be programmed and executed. With the addition of some critical com-

ponents, the system would actually be ready for use in a LOGO-based learning environment. The missing components include a working SD card driver and filesystem driver. Given a simple filesystem like FAT32 (the project's original target filesystem), such additions would not be too difficult. Without them, a ZNEO development environment would be required to update the LOGO files, which are currently stored as string literals in the C code.

2 Design Overview

2.1 LOGO

While LOGO is itself a Turing-complete programming language, it was never a goal of the project to implement the entire LOGO language. To do so would have added unnecessary complexity that would go unused in most situations, where preexisting languages like C would serve the purpose better. The major programming language constructs missing from the implementation of LOGO are variable assignment and flow control. It was originally planned to add a subset of LOGO flow control (loops) to reduce script length, but this was discarded when more critical issues arose. What is left is a basic set of imperative commands that allow for simple geometric control of the Turtle. With only these basic commands, a great range of interesting programs can be created. The commands can be broken up into three major groups, directional commands, distance commands, and basic flow control.

directional The directional commands available are `LEFT` and `RIGHT`, which provide counterclockwise and clockwise rotation, respectively. Each command has one mandatory argument, the number of degrees to rotate. Arguments are given in degrees, just as in LOGO, and executed to the accuracy permitted by the iRobot Create platform. Directional commands cause the Turtle to rotate the requested angle and then to immediately continue to the next command.

distance `FORWARD` is currently the only distance command implemented, as a functional `REVERSE` can be achieved through a 180 degree turn since the Create has a zero degree turning radius. Distance commands also have one mandatory argument. In LOGO, the unit is implicit, often pixels or logical units. In this implementation, the unit is millimeters, the native unit of the Create's commands. Distance commands cause the Turtle to drive the requested distance and then continue command execution.

flow control Only one command is included in the LOGO subset used for the project: the humble `END` command. `END` takes no arguments and simply informs the LOGO parser and Turtle that the script is over.

3 Hardware

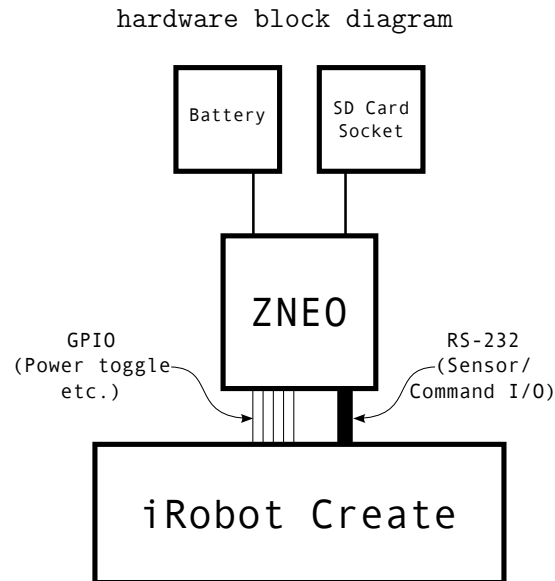
3.1 Materials

The project consists of the following materials:

1. ZNEO contest board
2. iRobot Create robotics platform
3. Create battery pack
4. 7-pin Mini-DIN to DE-9 serial cable
5. SD/MMC card socket (unimplemented)
6. Plug-in wiring
7. 9-Volt battery for mobile power of contest board
8. Blue Pikmin

3.2 Setup

The iRobot Create platform acts as the physical platform for the rest of the project. The ZNEO contest board sits atop the Create, along with the other minor components. The cargo bay located in the rear half of the Create provides an excellent location for excess cable slack. A large internal nickel-metal hydrate powers the Create. The Create then offers battery power to piggy-backed electronics by way of a cargo bay connector, but the voltage is not compatible with the contest board, so additional hardware would have been required. Instead, common 9-Volt batteries were used, as they are already compatible with the 9 volt DC input required for the contest board. Communication between ZNEO and Create occurs via RS-232 over a DE-9 to 7 pin Mini-DIN cable that goes from ZNEO to Create. The cargo bay connector on the Create also features GPIO pins, originally to be used for signalling command completion. Despite being able to trigger a manually-placed LED, the Create GPIO pins were unable to trigger functions on the ZNEO. To



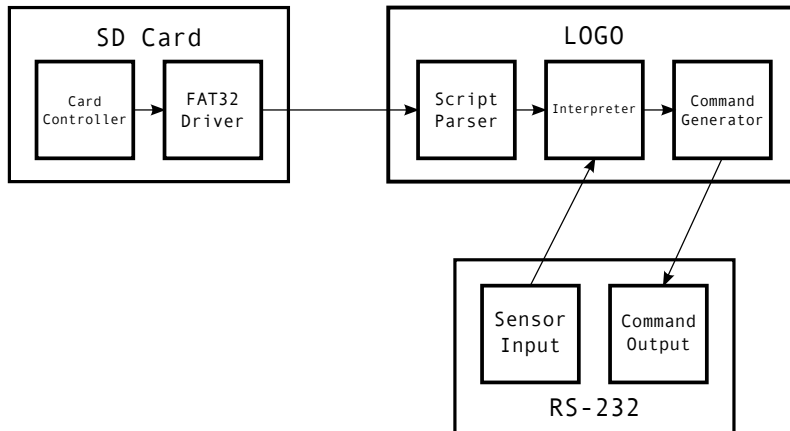
compensate, RS-232 communications were used as full duplex, allowing the Create to communicate back. Finally, the SD/MMC socket is connected to the contest board via plug-in wiring. Since both the SD format and the ZNEO have SPI communication built in, the socket is connected to the ZNEO SPI pins. Finally, The Blue Pikmin is installed in the cargo bay of the Create. The adorable nature of the Pikmin helps to alleviate human fears of Turtle-led robot uprising.

4 Software

4.1 Drivers

The components of the project all have well-defined boundaries, with well-defined interfaces between them. This greatly simplified creation of the drivers for ZNEO. For SD to ZNEO communication, SPI was used. Basic SPI support wasn't a problem, but the handshaking and formatting one layer above SPI proved a snagging point. SD and MMC cards have an interface used to read and write data as well as perform other actions such as query capabilities of the card. Each packet must also have a checksum to assure the integrity of the data. All of this was unnecessary for a project like the LOGO Turtle, and ended up hindering progress. For communication between the ZNEO and Create, RS-232 was used. Fortunately both ends of the connection support

software block diagram



RS-232 in hardware, so the only driver necessary was to configure the ports on the ZNEO for RS-232 transmission/reception.

4.2 LOGO Implementation

Processing of LOGO scripts occurs in two steps: parsing and interpretation. User-readable scripts are read into the parser, where they are broken down into operations and arguments. To accommodate this structure, each instruction is represented in C as a `struct` containing two fields: `op`, the operation, and `argument`. For the limited subset of LOGO implemented, the only arguments possible for commands are numeric, greatly simplifying the parsing operation. The lack of statement nesting also reduced complexity considerably, although the next step in development would be to add nesting for LOGO `repeat` statements. The (desirable) sacrifice of Turing-completeness overall made the implementation much manageable and robust.

Each LOGO line corresponds to one command `struct`. The commands are then placed sequentially into an array for the interpreter to read. For example, a simple LOGO program that traced an equilateral triangle with base length 100 millimeters would look like this:

```
FORWARD 100
RIGHT 60
FORWARD 100
RIGHT 60
FORWARD 100
```

END

This would then be parsed by the LOGO parser into a list of six 2-tuples as follows:

```
[ (0x01, 0x64),  
  (0x03, 0x3C),  
  (0x01, 0x64),  
  (0x03, 0x3C),  
  (0x01, 0x64),  
  (0x00, 0x00) ]
```

Each tuple contains an opcode indicating which operation to perform and the argument for that operation, or 0x00 if none is required. This list is then ready to be passed on to the LOGO interpreter for execution. The LOGO interpreter examines each command and constructs an iRobot Create Open Interface script (as described in the next subsection) and transmits it to the Create. The create executes the script and acknowledges its completion thereof. The acknowledgement from the Create triggers the LOGO interpreter to move on to the next instruction. When the END (null) instruction is reached, the interpreter shuts down and the program is complete.

4.3 iRobot Create Control

iRobot has created a serial command interface for the Create platform they call the 'iRobot Create Open Interface' or OI. The OI is similar to the format read by the LOGO interpreter of this project, consisting of strings of hexadecimal characters, only far more complex than any seen in the LOGO subset used. A leading opcode indicates which operation is to be performed, and a fixed number (depending on the command) of bytes follow as arguments. For example to make the Create rotate counterclockwise in place at a leisurely pace the following bitstring would be transmitted:

```
(0x89, 0x00, 0x0A, 0x00, 0x01)
```

The first byte, 0x89, indicates that a drive command is requested. The following two bytes, 0x00 and 0x0A indicate the speed, which is 10 mm/sec in terms of actual wheel rotation instead of angular velocity. The speed must be broken up into a high and low byte because the range of possible speeds exceeds the maximum representable value for a single 8-bit byte. The last two

bytes 0x00 and 0x01 are the turning radius. The two-byte value 0x0001 is a reserved value for turning in place counterclockwise.

To execute a series of commands at once, the OI also contains a `script` command. The `script` command is very useful for controlling the turtle, as it takes a considerable timing load off of the ZNEO. When a movement command is interpreted, a simple script is constructed. The Create is instructed to begin driving, either forward or rotating. Withing the same `script` instruction, the Create is also instructed to wait until it either drives the required distance for a `FORWARD` LOGO command or the required angle for a `LEFT` or `RIGHT` LOGO command. At the end of every script is a command for the Create to send a byte back to the ZNEO. This works as a sort of software interrupt, preventing the ZNEO from having to guess at how long to wait before the Create is finished or having to continuously poll the Create until it becomes responsive again after the script completes. By creating a short OI script for each LOGO instruction, the workload is equitably shared between the ZNEO and Create in executing LOGO scripts.

5 Retrospective

5.1 Improvements

Potential improvements abound for this project. Had implementation details not been so time-consuming, several would have been included in the project itself.

LOGO More of LOGO could be implemented. Specifically flow control, at least loops if not conditionals, would be a great boon to the language. For instance, the example LOGO script given in a previous section could be cut essentially in half by adding a `REPEAT 3` statement to handle the repetitive nature of tracing a triangle.

SD/MMC Support for memory cards would allow normal users unfamiliar with C or the ZNEO development environment to change the program quickly and easily. SD/MMC cards also come in sizes considerably larger than the entire ZNEO memory, allowing for massive LOGO scripts if a user were so inclined. Such an expansion would also require changes to the parser so the entire parsed script didn't remain in memory.

Filesystem A simple filesystem would make changing LOGO scripts as easy as dragging a text file for users, since many computers have SD/MMC

sockets, and USB sockets are inexpensive and widely supported. With additional modification to the main program, several LOGO scripts could even be loaded simultaneously and selected with the buttons on the contest board or by some other means.

5.2 Lessons

The biggest lesson in working on this project is not to underestimate bootstrapping tasks. Unexpectedly long tasks like attempting to work with the SD/MMC interface or deciding how to best control the Create took longer than expected and consumed valuable time that could have been used for one of the tantalizing improvements already listed.

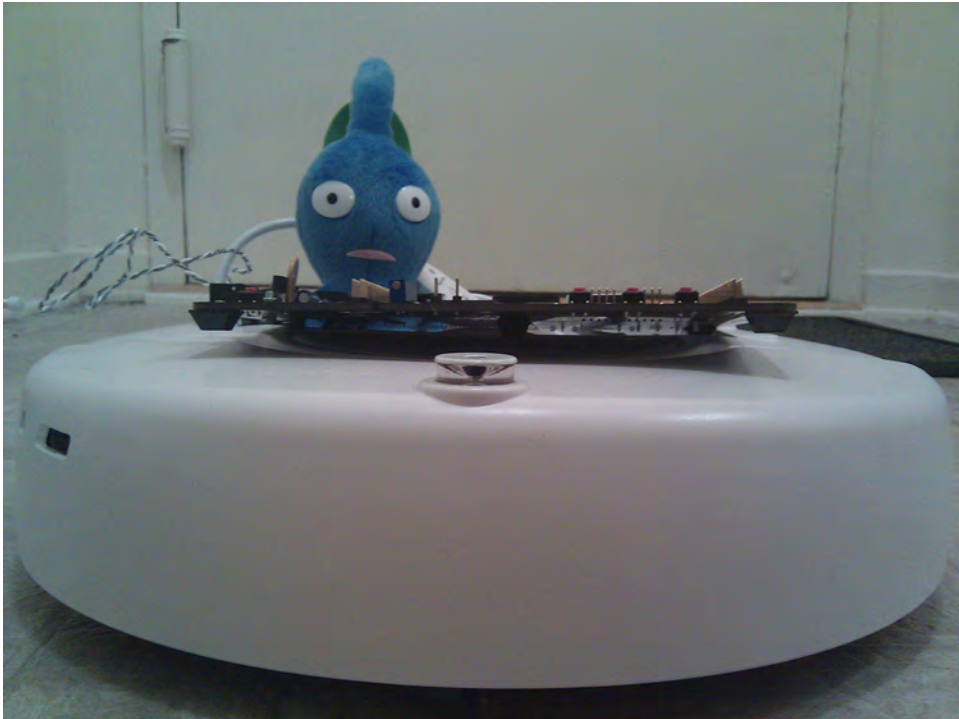
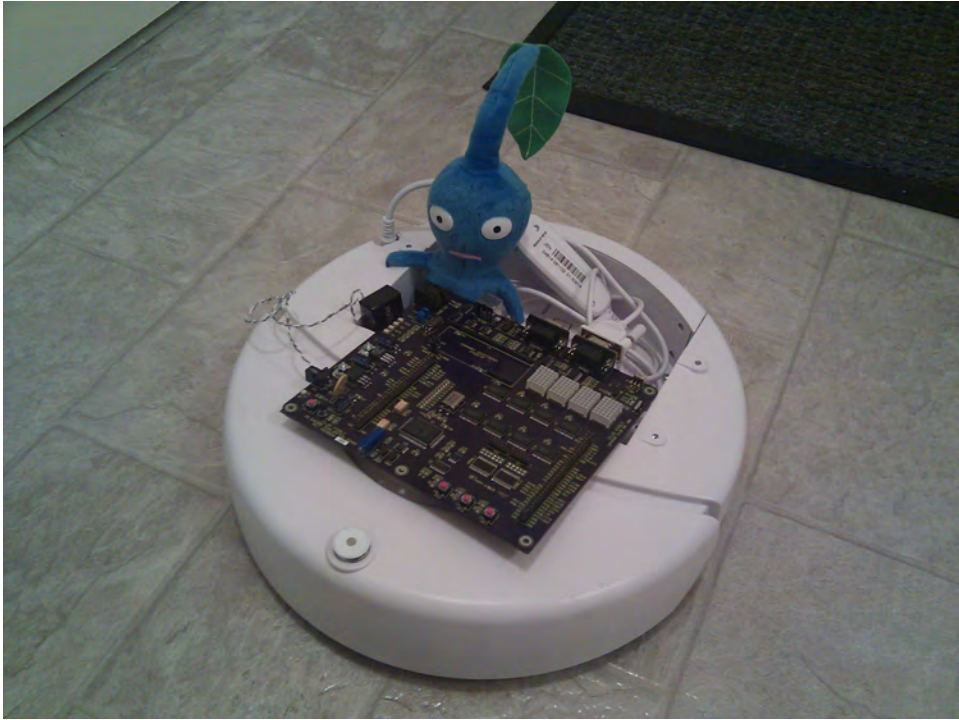
Another lesson is the importance of well-designed interfaces. A robust UART built into the ZNEO and a straightforward RS-232 interface for commanding the Create reduced the complexity of the overall project by a huge margin. Due to good, standardized interfaces, the components of this project were able to snap together like logical LEGO bricks.

Lastly, the flexibility of a system without an operating system turned out to be very liberating. Coming from only having worked with embedded systems with lightweight GNU/Linux systems, I had thought that working without an OS would be a daunting task. However, it turned out that when a simple task or group of tasks is the only goal for the system, the freedom and speed of direct control over the hardware is awesome.

6 Gallery



LOGO-like 'Turtle' control with iRobot Create and ZNEO



LOGO-like 'Turtle' control with iRobot Create and ZNEO

