

Yevgeniy Gindin

Embedded Systems: Final Project

Introduction

This project implants RCX to RCX communication via IR. In this implementation the sender RCX programs receiver RCX without any special software present on the receiver.

Requirements

- Two Lego Mindstorms kits (two RCX bricks are sufficient).
- PC set-up with LeJos (<http://lejos.sourceforge.net/>) and NQC (<http://brixc.sourceforge.net/nqc/>). RCX opcode reference (<http://graphics.stanford.edu/~kekoa/rcx/opcodes.html>)

Hardware / Software Description

A Hitachi H8 (H8) microcontroller is at the heart of RCX. H8 has 32K of external RAM. RCX can control three motors and accept input from three sensors. RCX uses a serial infrared port for communication.

RCX can be programmed in leJos, once leJos JVM is uploaded to RCX. Programs written in leJos are essentially Java programs. Not Quite C (NQC) is an alternative method to write programs for RCX. NQC is by far simplest to set-up as it uses standard LEGO firmware.

Project Description

Two LEGO robots were built. The sender robot (Competent) and receiver robot (Incompetent) each had two motors to enable movement. In addition, Competent used touch sensors to avoid obstacles.



Figure 1 – Incompetent robot.

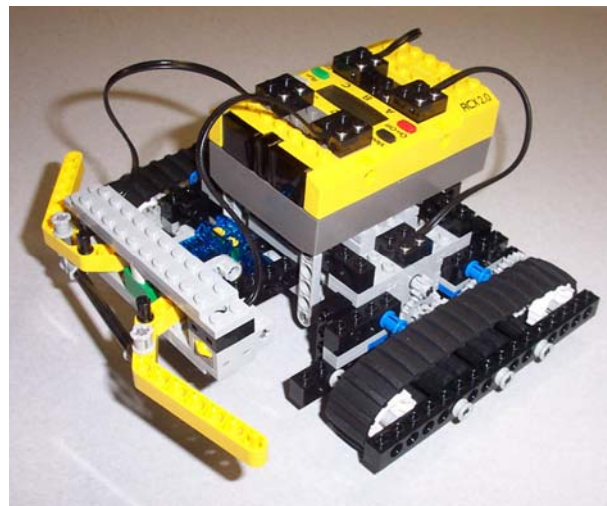


Figure 2 – Competent robot.

Competent was running a program compiled with leJos. Instructions that pertain to working with leJos can be found in the Requirements section of this document. Byte code transmitted by Competent was generated by compiling an nqc program using following command at command prompt: `nqc -L program.nqc`. The `-L` flag caused the NQC compiler to generate code listing.

Incompetent was loaded with standard LEGO firmware. There are no special software requirements for Incompetent. During presentation phase of this project, Incompetent will be running a routine that will simulate the sound of the theme song of Indiana Jones. Brick's Music Studio 1.5 was used to convert a .midi file to NQC instructions.

Discussion / Future Plans

The biggest challenge of the project was deciphering LEGO IR protocol. For each byte to be transmitted a packet has to be designed that includes a 3 byte header. The header is followed by the opcode, its complement, data byte, its complement and finally the checksum and its complement. In order to transmit a byte 3 the following has to be sent over the serial infrared: `0x55 0xff 0x00 0xf7 0x08 0x03 0xfc 0xfa 0x05` (where `f7` is the byte code for byte transmission and `0xfa` is the checksum of `0xf7` and `0x03`). The checksum is calculated by adding opcode and data bytes and taking modulo 8. This project used two opcodes: `0x25` to initiate task download and `0x45` to transmit data.

It was often difficult to get a reply from Incompetent RCX. Upon receiving byte `0x10`, Incompetent was to transmit byte `0xe7` or `0xef`. Rather than relying on Incompetent RCX to respond to 'alive' opcode, it was decided to repeatedly transmit 'download task' requests. This achieved desired effect while simplifying a lot of code.

In the future I plan to implement a method that will check if a transmission has been received successfully by an RCX.