

**The George Washington University
Department of Computer Science**

Final Project Report

for the

IRacer

By

**Michael Gaiman
&
Christopher Toombs**

CSCI 197
May 5th 2005

1. Description

The IRacer is an application to allow the remote control of an RCX-equipped vehicle from a J2SE enabled device using Not Quite C (NQC). Both the controller and the vehicle must support infrared (IR) communication, as this is the medium for command passing. The user issues commands through the keyboard arrow keys; the vehicle supports the following movements: forward, backwards, left, right, and any non-orthogonal combination therein.

2. Vehicle Design

The IRacer vehicle design is shown in Figure 1. It utilizes a modified Lego® Roverbot driving base which has been modified to support a tricycle design for improved turning capabilities. Furthermore, we suspend the RCX 2.0 above the base on an incline with the lowest point closer to the base at the forward end of the vehicle. The incline elevates the IR receiver which allows for greater visibility, which improves the relative angle of data transmission. Due to the suspension of the RCX 2.0, the vehicle is top-heavy and prone to tip during a rapid backward deceleration. The design prevents the vehicle from tipping over by the addition of a stabilizing tail protruding approximately two inches from the driving base in the rear of the vehicle.

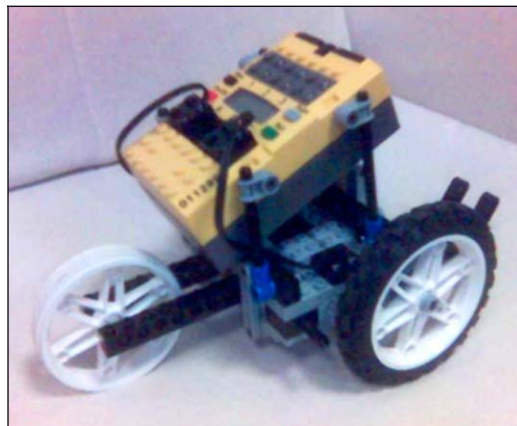


Figure 1 - RCX vehicle design.

3. Hardware

3.1. IRacer Controller

The IRacer controller requires several pieces of hardware for operation. These devices interact as illustrated in Figure 2. The user issues commands through keyboard depressions. The keystrokes generate interrupts and are buffered until read by the J2SE device. The software within this device converts the keystroke signal into IRacer protocol commands that the vehicle translator will understand. These signals then propagate to the infrared transmitter for transmission.

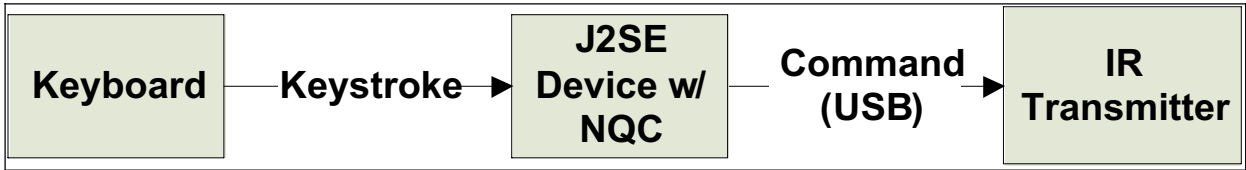


Figure 2 - IRacer controller hardware.

3.2 IRacer Vehicle

The IRacer vehicle requires several pieces of hardware as well. These pieces and the flow of data between them are shown in Figure 3. The RCX IR Receiver captures the infrared message from the controller IR Transmitter. Next, the digitized signal propagates to the RCX 2.0 “Brick”. At this point, the IRacer protocol commands are extracted from the signal. Based upon the interpretation of the received command, the Brick issues commands to one of three devices. Two of the devices are servos which control the movement of the vehicle. The manner in which signals are propagated to these servos determines the movement. For instance, a forward signal to the left servo combined with no signal to the right servo results in the vehicle turning right. Similarly, a forward signal to both moves the vehicle forward, while a backward signal to both moves the vehicle backward. Finally, the Brick may issue signals to the speaker which will provide audible feedback to the controller. This signal is represented by an integer which indicates the frequency to be generated by the speaker.

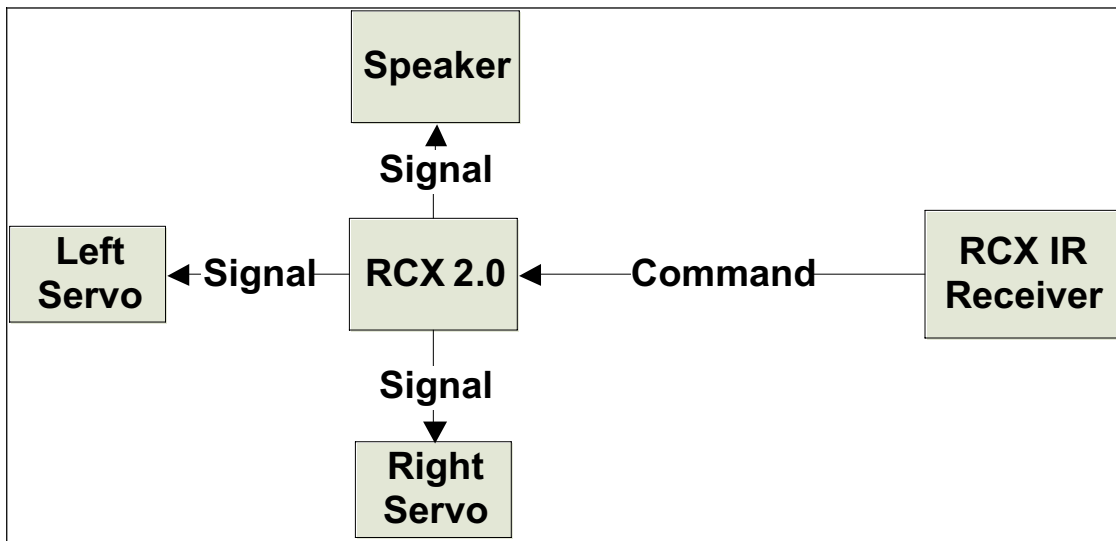


Figure 3 - IRacer vehicle hardware.

4. Software

4.1 IRacer Controller

The IRacer controller application makes use of three software modules, two of which were specifically implemented for this device. The data flow between these modules is illustrated in **Figure 4**. These modules reside on the J2SE device from **Figure 2**. The Graphical User Interface (“GUI”) is actually not terribly graphical. A small window is displayed on the screen while the application is active on the device. This application receives the keystrokes from the user. This data then propagates to the Sender Translator module which interprets the keystrokes and issues the appropriate command. The command is based on the IRacer communications protocol. The Sender Translator then issues this command to the NQC Send module which will instruct the IR Transmitter from **Figure 2** to communicate the command.

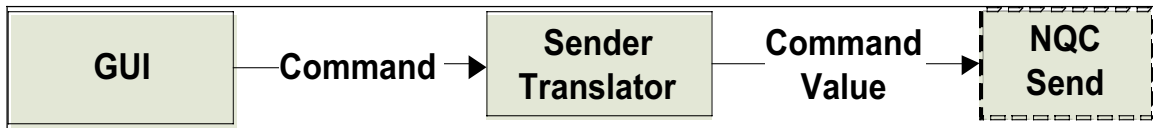


Figure 4 - IRacer controller software.

4.2 IRacer Vehicle

The IRacer vehicle also contains four major software modules to receive and process the command from the controller. The data flow between these modules is shown in **Figure 5**. These modules reside on the RCX 2.0 hardware device from **Figure 3**. The NQC Receive module captures the message received from the RCX IR Receiver from the IR vehicle hardware and propagates this data to the Receiver module. The Receiver module resets the Watchdog Timer sub-module upon receiving a message; the timer is set to timeout at 1.5 seconds. This is used to stop the vehicle if contact between the controller and the vehicle is lost. The Translator sub-module receives the message and interprets its contents based upon the IRacer communications protocol, and generates the appropriate instruction. Depending on the specific instruction, either the NQC Servo Control module or the NQC Speaker control module is called. The NQC Servo Control module controls both the right and left servos from the IRacer vehicle hardware (**Figure 3**), while the NQC Speaker Control module controls the speaker.

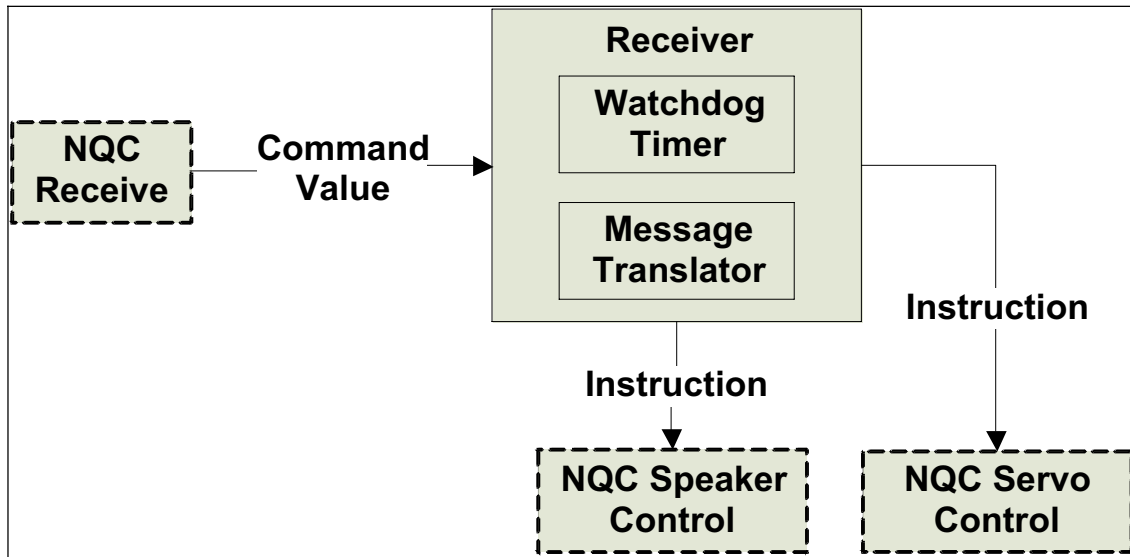


Figure 5 - IRacer vehicle software.

5. IRacer Communications Protocol

Due to the limited number of possible commands, the entire set of commands can, and is, encapsulated in a byte of data (8 bits). The mapping of logical commands to their byte encoding adheres to the following rules:

Table 1 - IRacer communications protocol summary.

Command	Command Code	Command Value
Forward	(TG_FWD)	0x01
Backward	(TG_REV)	0x02
Turn Left	(TG_LFT)	0x04
Turn Right	(TG_RGT)	0x08
Sound	(TG_SND)	0x10
Stop	(TG_STP)	0x20

Note that the choice of command values allows for the application of the bit-wise OR operator so that multiple commands may be transmitted in one message. Hence, the extraction of commands from the message relies on application of the bit-wise AND operation on the value with respect to the Command Code.

6. Changes from Initial Proposal

Our initial proposal was to implement the communications over Wi-Fi. Control of the vehicle would be conducted using the buttons from the Zilog Z8 Encore. This

proved to be infeasible because we could not find the appropriate hardware. That is, we required a device with an implementation of an IP stack that could interface with the Zilog Z8 board as well as the RCX 2.0 within our financial constraints. We were unable to do so, and thus were forced to modify the project.

The second iteration of the project replaced the Wi-Fi component with radio frequency. User control would still occur at the Z8, but commands would be sent as radio signals to a receiver on the RCX 2.0. Fortunately, we were able to find compatible hardware within our budget and purchased the items. We received a MINDSTORM compatible breadboard and connectors within a day. However, a second vendor from whom we ordered the receiver and transmitter proved less reliable. After waiting a number of days, we finally purchased the items from a different vendor. This left little time to construct the hardware. We decided to redesign the project again so that we may turn in a complete, working project that is simpler than the initial proposal rather than a slightly more complex but ill-functioning system. The result of this redesign is the current project.

7. Specifications

7.1 IRacer Controller

The IRacer controller requires a graphical windowing environment with an installation of Java 1.4.2 or above. Furthermore, the controller must have an installation of Not Quite C version 3.1.r1 or above to interact with the IR transmitter.

7.2 IRacer Vehicle

The IRacer vehicle has only been tested using the RCX 2.0 and we require this device to ensure proper operation. It additionally requires that the two servos are configured for driving as opposed to turning. This is because the IRacer turning implementations operate by powering only one servo during the duration of the turn. By default, Port A is used for the left servo, and Port C is used for the right servo. This can be changed by modifying TG_ACT_L and TG_ACT_R constants in Drive2.nqc.

8. Compiling & Execution Instruction

8.1 IRacer Controller

The controller is compiled by issuing the following command in the directory in which IRacer.java resides (“\$” indicates a command-line prompt – do not type the “\$”):

```
$ javac IRacer.java
```

After compilation of the controller software, the application may be executed via the following command.

```
$ java IRacer [path to NQC application [RCX_PORT]]
```

The parameters of the execution are both optional. The first parameter, the path to the NQC application specifies the location of the NQC application, which is required for proper execution. When this parameter is left blank, the default behavior of the application is to look in the current directory. The second parameter, RCX_PORT, specifies the port on which the IR transmitter is located. If this parameter is left blank, the default behavior is to set the parameter to USB. See NQC documentation for more information. This parameter may only exist if the first one is listed.

8.2 IRacer Vehicle

We used the MacNQC IDE to transmit the vehicle software to the vehicle. However, the software for the IRacer vehicle can also be transmitted to the RCX 2.0 by BricxCC or via an NQC command-line application. Consult the appropriate documentation for more details. Verify that the vehicle is on and operational prior to the transmission of the application data.

Once the data has been sent to the RCX, the application can be launched by navigating to the correct program through the Prgm button the RCX Brick, which is the grey button in Figure 6. Once the correct application is selected, press the green “Run” button to begin execution. The RCX will emit a brief series of tones at this point if this has been done successfully.

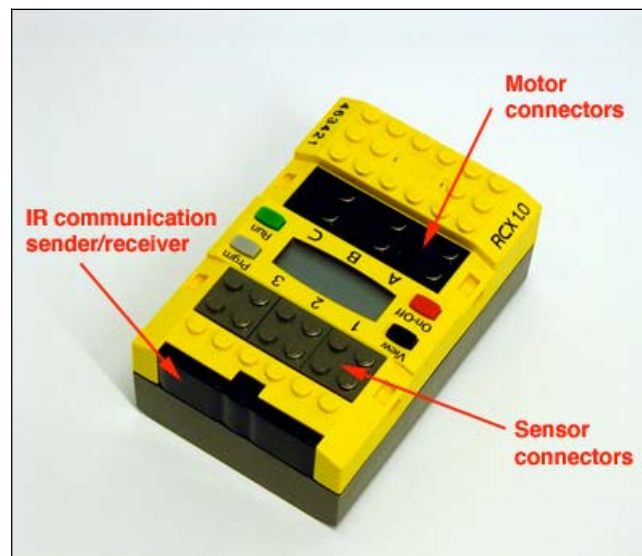


Figure 6 - Location of RCX inputs, outputs and buttons.

9. Development Summary

Development began with the structural design and construction of the vehicle. Several models were used prior to the one used in the completed project. Each design was tested by driving the vehicle via wired communication, utilizing the touch sensors supplied in the Lego® MINDSTORMS kit. The initial design had four wheels; this proved to limit the turning ability of the vehicle. Once the tricycle design was selected, we noted the limited angle at which a controller could communicate with the device. This led to the development of the current inclined design for the RCX brick. However, this modification made the vehicle top-heavy and prone to tipping, at which point the tail stabilizer was added.

Parallel to construction of the vehicle, development began on the software components of the application. First, software for the vehicle was built which would drive the vehicle in a based upon input from the RCX touch sensors. This was used to ensure the code would drive the servos and to test structural designs. Next, development focused on the vehicle's Receiver module, which did not initially include the watchdog timer component. The module was tested through command-line invocation of NQC. Once infrared communication was established, development moved to the controller sender module. The initial version simply verified the transmission of data via infrared to the vehicle.

Next, the protocol went through several iterations. Initially, communication used a flood-based protocol which was undesirable as commands rapidly became out of sync. This was due to latency in the infrared system which was slower than the processing of messages by the Java application. We switched to a system in which messages are only sent as fast as the infrared system can handle; keyboard events may be ignored in this case. In dealing with latency issues, the watchdog timer was developed and added to the system. This has now proved beneficial in its ability to terminate execution of the current movement of the vehicle when it can no longer detect a command.

Finally, development focused on the controller user interface through which a user issues commands to the vehicle. This was perhaps the easiest part of the project.