The Human Touch with `iRemote`

Paul Newman

September 28, 2007

Abstract

This document will give you a description of how to use and configure `iRemote` to inject commands into a community and also to query the status of certain commonly used variables from a terminal.

1 Manual Control - `iRemote`

`iRemote` was designed to be a control terminal for a deployed vehicle. It is really nothing more than a long `switch` statement based on characters input from the keyboard. One of its many functions is to allow remote control of the actuators of the vehicle. This is an invaluable asset for land and sub-sea vehicles alike. The application is multithreaded. The primary thread blocks on a read of keyboard input. When a character is pressed some action is taken - for example publishing a new value for `DESIRED_THRUST` . The fact that `iRemote` can take control of a real vehicle presents a safety problem. What if the human controller walks away or even worse the vehicle moves out of communication range (eg a submarine dives) and the console is not available? To prevent the last issued actuator command being carried out *ad-infinitum* a secondary in thread `iRemote` prompts the user to hit an acknowledge key (') at least every 15 seconds. If the human driver does not respond the all actuators are set to the zero position.[1]

1.1 Summary of Functionality

The following (not exhaustive) list describes some of the online functionality that `iRemote` provides:

[1] The actuation driver in iActuation will also shut down all motors if it does not receive control commands for an extended period of time. In the Bluefin vehicle the driver class within iActuation sends a message to the Janitor processes which resets a watchdog on the power management board - keeping the tail cone powered.

Function	Key	Comment
Reload Mission File	R	Tells `pHelm` to rebuild its tasks
Restart Navigation	V	Tells `pNav` to reboot all navigation filters
Restart Logger	G	Tells `pLogger` to begin recording to a new set of log files
Begin Mission	O	Instructs `pHelm` to go online
Halt Mission	O or *space*	`pHelm` goes offline and `iRemote` takes control immediately.
Navigation Summary	*	Prints a summary of salient navigation information
Rudder Left/Right	N,M	Steer control
Elevator Up/Down	P,L	Pitch control
Thrust Up/Down	A,Z	Throttle control (+ shift gives 100 percent)
Stop	*space*	Immediate zero of all degrees of freedom
Fetch DB	F	Prints a summary of the contents of the entire `MOOSDB`
CustomKey	$[0 \rightarrow 9]$	The numeric keys can be made (via `iRemote` s configuration block) to publish any named variable with a specified value. In the example configuration block below (Figure 1), pressing key "2" will cause `iRemote` to write the variable `JANITOR_SWITCH` with the string value (quotes) ACTUATION::OFF
CustomSummary	+	The configuration block allows a custom summary to be built consisting of any variable names used within the system. `iRemote` subscribes to this data and prints its current value when requested.
CustomJournal	$[0 \rightarrow 9]$	Similar to *CustomSummary* but instead of keeping the most recently published variable it keeps a history of values. Each Journal can be bound to a numeric key. In the example below pressing key "6" will show the past 10 values of `DESIRED_RUDDER` with every delta captured as the capture time is 0.

1.2 Informing the Pilot

In most missions `iRemote` is the only interface the vehicle pilot has with the vehicle. Clearly then a method is needed by which *important* information can be sent to the `iRemote` console from any process. The `CMOOSApp` member function `MOOSDebugWrite` achieves this by issuing a notification on a variable watched by `iRemote` . Such messages are displayed on the iRemote console

```
//////////////////////////////////////////////
// iRemote config block
ProcessConfig = iRemote
{
    CustomJournal = Name = DESIRED_RUDDER, Key =6, History = 10, Period = 0
    CustomSummary = DESIRED_THRUST
    CustomKey = 2 : JANITOR_SWITCH @ "ACTUATION:OFF"
}
```

Figure 1: A small configuration block for iRemote showing typical usage of the `CustomX` commands

at run time along with the process making the announcement. Note that this name is somewhat unfortunate as this function should not be used for debugging - it is a run-time thing. It is frustrating to have a cornucopia of messages flashing on the screen during a mission the content of which is meaningless to the pilot. Typical uses of this functionality would be a very occasional summary of navigation status and system level warning messages - for example notification of unexpected mission task termination.