Novint Falcon Robot Raconteur Service

Contents

1 Reading This Document

This document is intended for a reader that has a good understanding of using Robot Raconteur (RR) `http://robotraconteur.com` and Microsoft Visual C/C++ (MSVC) Projects built using Microsoft Visual Studio. This document does detail all of the required steps to set up and build the Novint Falcon RR Service, so a user with little experience in these topics need not worry.

Little to no prior knowledge of Haptics and Haptic Feedback devices are required to use the Novint Falcon RR service. However, a more experienced user wishing to expand this service or use more haptic related feedback may want to refer to Novint's *HDAL Programmer's Guide* included in the `doc` folder of the HDAL SDK installation.

This document includes instructions for installing and setting up the Novint Falcon and the Falcon Robot Raconteur Service on a new computer. For troubleshooting purposes one may need to refer to Novint's documentation on setting up the Falcon, or to Robot Raconteurs documentation on using RR with C++, as this document may not cover all cases. Sample MATLAB and Python clients are included in the GitHub repository if the user simply wishes to use the service. Running the service does require building the source code as there are currently no built binaries included. Note that due to several Windows only dependencies, the RR Service is currently available for Windows computers only. The service has been tested using Windows 7 and MSVC 11.0/12.0.

2 Novint Falcon

The Novint Falcon is 3-DOF haptic device made by Novint for use in the gaming industry as a joystick controller. Haptic devices, such as the Falcon, differ from traditional joysticks in that they provide the user with force feedback that allows the user to interact using their sense of touch. (After setting up the Falcon I highly recommend installing and playing with the Novint Falcon Tutorial). The Novint Falcon RR Service makes it easy to use the Falcon in any programming language supported by Robot Raconteur by exposing most of the Falcon's main functionality. For more information on the Falcon, check out `http://www.novint.com/index.php/products/novintfalcon`

3 Required Downloads

3.1 Project Source Code

The source code for this project is housed in a GitHub repository located at `http://github.com/rpiRobotics/NovintFalcon`. This repository should be cloned to your computer or the .zip file of the project should be downloaded from this site.

3.2 Novint Falcon Drivers

In order to use the falcon on your computer the correct drivers must be installed. The drivers for the Novint Falcon can be downloaded from the Novint Falcon Download Page (`http://www.novint.com/index.php/downloads`).

3.3 Project Dependencies

The Novint Falcon RR Service has several dependencies that must also be downloaded to your computer in order to build and run the service. The following should be downloaded somewhere on your computer.

3.3.1 Robot Raconteur C++ SDK

The RR C++ SDK can be downloaded from the RR website at `http://www.robotraconteur.com/download`. The service has been tested using RR Version 0.5-testing. Download the C++ SDK specific to your version of MSVC.

3.3.2 Boost 1.55.0

As mentioned on the Robot Raconteur page, the RR C++ SDK depends on the Boost C++ Libraries. Boost 1.55.0 can be downloaded from `http://sourceforge.net/projects/boost/files/boost/1.55.0/`

3.3.3 Novint HDAL SDK

The Novint HDAL SDK allows programmers to write programs specifically for use with the Novint Falcon. The SDK can be downloaded from Novint's downloads page at `http://www.novint.com/index.php/downloads`.

The SDK currently supports Windows machines only, hence the Falcon Robot Raconteur Service's limitation to Windows. However, an alternative open source cross-platform SDK called *libnifalcon* does claim to provide an alternative to the HDAL SDK. If a future user wishes to run the service on a different platform this may be an option to explore, but be warned the libraries may not be easy to build. More information can be found at `http://qdot.github.io/libnifalcon/`.

4 Installation

Several more steps must be taken to set up your computer and these newly downloaded dependencies before the RR Service project can be built.

4.1 Building Boost

In this section I will detail how to build the required Boost dependencies specifically for use with Robot Raconteur. For more information on the Boost libraries, including how to do a full installation, you may want to check out the following resources:

- `http://www.boost.org/doc/libs/1_55_0/doc/html/bbv2/installation.html`

- http://www.boost.org/doc/libs/1_55_0/more/getting_started/windows.html
- http://www.boost.org/doc/libs/1_55_0/

The instructions in this section are taken from the *Robot Raconteur using C++* guide available for download from https://robotraconteur.com/documentation/. The relevant instructions are repeated here for the user's convenience.

On Windows, Boost is built using the "Visual Studio Command Prompt" or "Windows SDK Command Prompt" that matches the version of Visual Studio/MSVC being used. For Visual Studio 2010 or MSVC 10.0, use the following commands in the unzipped boost directory:

```
> bootstrap
> b2 --with-date_time --with-thread --with-system --with-regex
    --with-filesystem --with-chrono --stagedir=stage32
    --toolset=msvc-10.0
```

For Visual Studio 2012 or MSVC 11.0, run:

```
> bootstrap
> b2 --with-date_time --with-thread --with-system --with-regex
    --with-filesystem --with-chrono --stagedir=stage32
    --toolset=msvc-11.0
```

For Visual Studio 2013 or MSVC 12.0, run:

```
> bootstrap
> b2 --with-date_time --with-thread --with-system --with-regex
    --with-filesystem --with-chrono --stagedir=stage32
    --toolset=msvc-12.0
```

The 32-bit libraries will be created in the `stage32` directory.

4.2 Setting up HDAL SDK

The HDAL SDK installer should be run, and the packages installed to wherever is easiest for the user.

During the installation, the installer will set some recommended environment variables. These environment variables should be checked before continuing as they may cause problems later. The `NOVINT_DEVICE_SUPPORT` environment variable should be set to your install directory for the HDAL SDK. The install should also add a new entry to your system `PATH`. `PATH` Should begin with "`.;XXXX\bin;`", where `XXXX` has the same value as `NOVINT_DEVICE_SUPPORT`. Note the leading "."; this allows you to put trial versions of DLLs into the application's folder for testing without disturbing the normal DLLs used for all other applications.

Novint uses the `NOVINT_DEVICE_SUPPORT` environment variable to determine which libraries to use for the Falcon. If the falcon is being used as a

joystick for a supported computer game the environment variable should be set to the `Novint\Falcon\HDAL` folder that was created when the Novint Drivers were installed (probably inside of `Program Files(x86)` on a Windows computer). If the falcon is being used by a developer with the HDAL SDK (as we will be using it) it should be set to the HDAL SDK folder.

If the variable does not appear to be created correctly or the user does not wish to use the environment variables, they are not necessary. The full path to the HDAL SDK folder may be used later on when setting up the Visual Studio project instead of using the environment variable. As mentioned before, the required DLLs (inside the `<HDAL_SDK_PATH>\bin` folder) may also just be copied to the same folder as the executable in order to run the program. The rest of this document will assume that these environment variables are correctly set.

5 Configuring the VS Project

If you have followed the previous sections your computer should have everything it needs to build and run the Project. Now the Project itself must be configured to find all of these dependencies on your local machine. In the following directions, items in angle brackets (< >) are a description meant to be filled in with the value specific to your local machine. Items in bold are Visual Studio options.

- Open the Visual Studio Solution included in the GitHub Repository: `<NovintFalcon>‣Service‣Falcon_RRService.sln`.
- Open the Project Properties
  Project » Properties
- Under the Configuration Properties » General tab.
  - Platform Toolset ⇒ set to your version of MSVC.
  - Character Set ⇒ **Use Multi-Byte Character Set**
- Under the C/C++ » General tab.
  - Add the following to Additional Include Directories

    ```
    <RR_CPP_PATH>\include;
    <boost_1_55_0>;
    $(NOVINT_DEVICE_SUPPORT)\include;
    ```

- Under the Linker » General tab
  - Add the following to Additional Library Directories

```
<boost_1_55_0>stage32\lib;
<boost_1_55_0>\libs;
<RR_CPP_PATH>\lib\Release;
$(NOVINT_DEVICE_SUPPORT)\lib;
```

- Under the Linker > Input tab
  - Add the following to Additional Dependencies

```
Rpcrt4.lib;
ws2_32.lib;
IPHLPAPI.lib;
RobotRaconteur2.lib;
hdl.lib;
```

The project should now be correctly configured to build on your local machine. Note that due to a limitation in the HDAL SDK the project can be built in **Release** mode only.

6 Running the Service

There is not much more to running the service than starting it from the Visual Studio debugger or running the executable once the VS Project has been built. This section serves more as a troubleshooting guide to document some of the reasons the service may not run correctly. When running the service please ensure the following:

- The Falcon is plugged in both to your computer and to a wall socket. The lights on the Falcon will come on if it is only connected to a computer but they will be dim and the Falcon may not be able to connect.

- Your computer knows where to find `hdl.dll, dhdlc.dll,dhdlcDriver.dll` and any other drivers it may need.
  These drivers are located in the `<HDAL_SDK>\bin` folder which should be on your system `PATH`. If the HDAL SDK install did not correctly set your `PATH`, or you do not want to change your `PATH` you may copy the contents of `<HDAL_SDK>\bin` into the directory containing your executable and try running the program again.

- The lights on the Falcon are blue.
  If the lights on the front of the Falcon are red the device needs to be re-homed. When the service starts up it should automatically check if the Falcon needs to be re-homed and prompt the user to re-home the device. Re-homing is accomplished by pulling the end effector of the joystick all the way out, and then pushing it back in towards the

base at which time the light should change from red to blue. If the light turns red during use, try performing this re-homing operation and consider restarting the service.

6.1 Connecting to the Service

The connection to the Robot Raconteur service is made over TCP. The connection address is displayed in the Terminal when you launch the service. The default port for the Falcon is 2354. Currently there is no command line interface option to set a different port. However, if you have built the service from source you may change the port in the `Main.cpp` file.

6.2 Available Properties

6.2.1 position

The `position` property returns an array of three (3) doubles specifying the X, Y, and Z positions respectively of the Falcon's end effector. Note that these positions are raw positions and have not been scaled at all.
This is a **READ ONLY** property.

6.2.2 button_status

The `button_status` property returns a single integer that represents the bitwise status of all of the buttons. The bits for each button are

$$[c|l|t|r]$$

for the center, left, top, and right buttons respectively. A 0 in a buttons bit represents unpressed, while a 1 represents pressed.
This is a **READ ONLY** property.

6.2.3 controller_input

The `controller_input` property returns a struct that has six (6) fields, all integers.

- `positionX`
- `positionY`
- `positionZ`
- `center_button`
- `left_button`
- `right_button`

- `top_button`

All position values are scaled to a number between 0 and 10000 relative to the deadzone if it is enabled. Each button also has its own field where a 0 is unpressed and a 1 is pressed.
This is a **READ ONLY** property

6.2.4 deadzone_enabled

The `deadzone_enabled` property specifies whether a deadzone should be implemented when scaling the position values returned in `controller_input`. The value of this property is returned as an integer where a 0 means the deadzone **is not** enabled and a 1 means the deadzone **is** enabled.
This property can be both read and set.

6.2.5 deadzone_size

The `deadzone_size` property specifies what percentage of the total workspace the deadzone should take up around the origin. The default value is 20 (20%). The number should be an integer between 0 and 100.
This property can be both read and set.

6.3 Available Functions

6.3.1 setForce

The `setForce` function takes as input an array of doubles of size 3 specifying the desired forces the Falcon should apply in the X, Y, and Z directions respectively. The forces should be given in units of Newtons [N].

7 The Code Explained

The code for the Novint Falcon RR Service is broken down into a hierarchy of several different layers. At the top is the actual Robot Raconteur service calls that start up the Robot Raconteur Service and provide the user interface. This layer is defined in the `Main.cpp` file.

For RR to interface to work with the Falcon it must have a definition of the Falcon Service being implemented. This service definition is included in the `falcon_service.robodef` file. This service must then be translated into C++ 'thunk' code. This thunk code is included in the project as the `falcon_service.h`, `falcon_service_stubskel.h`, and `falcon_service_stubskel.cpp` files.

`falcon_service.h` specifically defines the '`Falcon`' and '`ControllerInput`' interface classes that must be implemented. These classes are implemented in `falcon_impl.h` and `falcon_impl.cpp` respectively. You may notice that

the declarations in `falcon_service.h` and `falcon_impl.h` are nearly identical.

The implementation layer implements the service, interacting with the Falcon/Haptic layer below it and the RR Service layer above it. It also performs the post-processing of the raw data from the Falcon where necessary. The `Vector3.h` and `Vector3.cpp` files specify a helper class, the `Vector3` class, which defines a vector and several operations that can be done on vectors. The deadzone and normalized position operations make use of this class by treating the returned raw position values as a directional vector from the origin and then normalizing it.

The final layer is the actual Falcon/Haptic layer implemented in the `Haptics.h` and `Haptics.cpp` files. These files call functions from the HDAL SDK that then communicates directly with the Falcon.