

FRC CAN Bus Control Via the Black Jaguar

Mike Anderson (manderson13@cox.net)

Herndon High School

Herndon, VA

Epsilon Delta Team 116

February 10, 2010

Overview

In the search for ways to make robot cabling simpler and more robust, Team 116, like many others, spent some time researching the use of the controller area network (CAN) bus capability that is present in the TI (formerly Luminary Micro) Jaguar motor controllers. 2010 is the first year where the CAN bus is legal for competition. Unfortunately, since this is the first year where it can be used, the level of experience with the CAN bus is still relatively low.

In trying to better understand the use and deployment of CAN bus control, there are a number of disparate sources of information. Much to our dismay, there is also a lot of conflicting information that can lead to confusion. The purpose of this write up is to try to bring all of these pieces together into a single document that can serve as a guide to others as they examine CAN bus as an alternative to the traditional PWM wiring and motor control approach.

Hardware Set up

Currently, there are two different techniques for interfacing the cRIO controller to the CAN bus. One is using the 2CAN Ethernet-to-CAN from Cross the Road Electronics (<http://andymark.biz/am-0545.html>). Since the 2CAN is a 2-port Ethernet switch that interfaces to CAN bus, you can use it to talk with the CAN bus by plugging into Ethernet port 2 on the cRIO and plugging the Axis camera into the other port on the 2CAN. There is also a software plug-in available for the 2CAN (FRC_2CANPlugIn.out) as well as example code that shows how to use the 2CAN for motor control. Look at the 2CAN user manual for more details on how to use the 2CAN as a CAN gateway for motor control.

As interesting as the 2CAN unit may be, the subject of this write-up is the use of TI's black jaguar motor controller to interface to the CAN bus. The black jaguar contains a serial to CAN bus interface. Because the CAN bus is typically run at 1 Mbps, the serial port can represent a bit of a bottleneck at it's 115Kbps data rate. However, due to the nature of the FRC robots and a special CAN mode known as synchronous mode, the speed of the black jaguar interface should not be much of an issue. Figure 1 shows the typical hook up for the black jaguar controller.

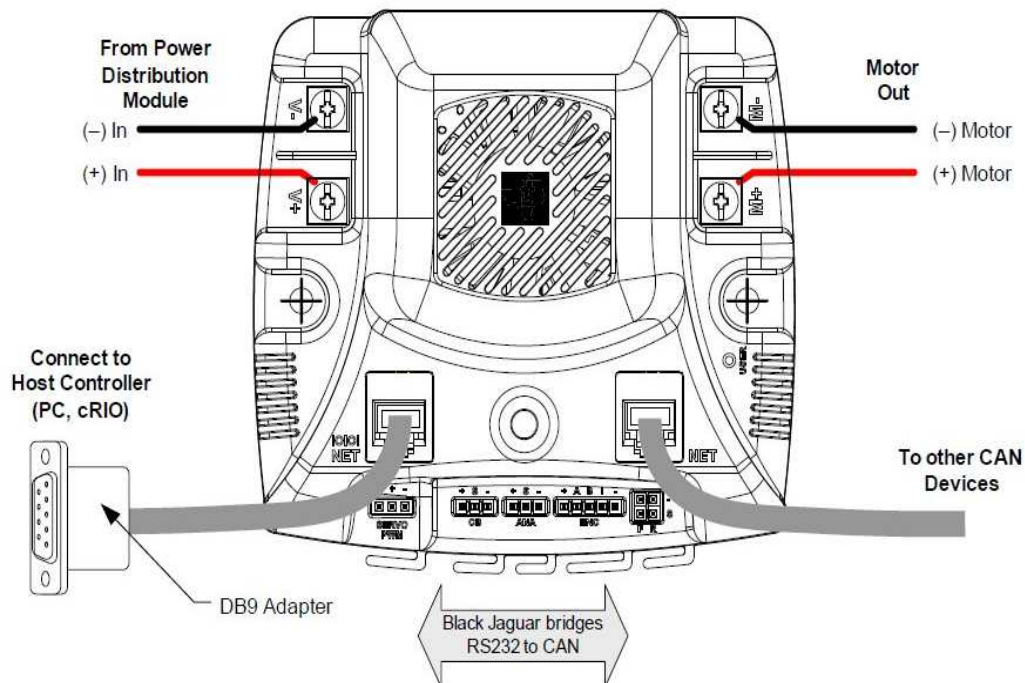


Figure 1. MDL-BDC24 (Black Jaguar) RS-232 to CAN Bridge
(Source: TI)

The construction of the CAN bus cables is your first challenge. The CAN bus connectors being used are RJ-12 6P6C (6-pin, 6 conductor) type connectors that are readily available at Radio Shack, Home Depot or other home improvement stores. I got mine as well as the 6P6C crimp tool from Lowes. You cannot use RJ-45 Ethernet crimping tools or connectors. They are too wide (8P8C) to fit into the connectors on the jaguars. Also, you cannot use typical telephone cords. First, many telephone cords are only 2-conductors and second, the CAN bus requires a “straight through” cable where the modular plugs are reversed as shown in Figure 2.

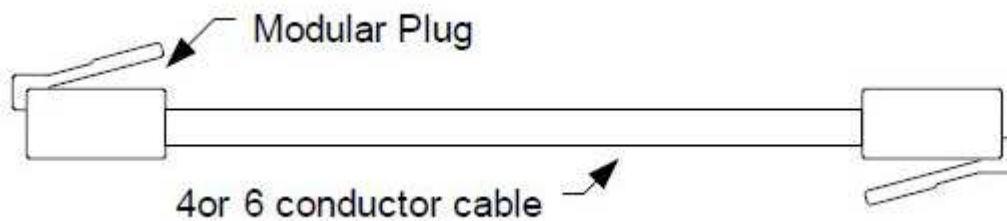


Figure 2. CAN Bus Cable w/ Reversed Connectors
(Source: TI)

The cable pin assignments generally follow a color-coding scheme as seen in Figure 3.

RJ12	Wire Color	Jaguar Use	PC Use	DB9 Pin
1	White	RXD	TXD	-
2	Black	-	-	-
3	Red	CAN H	-	-
4	Green	CAN L	-	-
5	Yellow	GND	GND	5
6	Blue	TXD	RXD	2

Figure 3. Standard Telephone Wire Coloring for RJ12
(Source: TI)

Using RJ12 connectors (6P6C) will have the 4 wires in the center with pins 1 and 6 empty for the typical CAN bus cable. RJ11 connectors (6P4C) will only have the 4 inner conductors and be missing pins 1 & 6. If you happen to have some of the newer cabling laying around that doesn't use the older color scheme, you'll find the newer color scheme is as follows:

Old	New
White/Orange	White with Brown Stripe
Black	Orange with White Stripe
Red	Blue with White Stripe
Green	White with Blue Stripe
Yellow	White with Orange Stripe
Blue	Brown with White Stripe

The orientation of the RJ-12 connector is shown in Figure 4. Remember to have one connector tab up and one connector tab down when you make this cable!

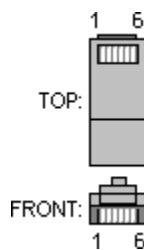


Figure 4. RJ-12 Connector Orientation and Pin Out
(Source: <http://pinouts.ru>)

Next, wiring the RS-232 connector has its own challenges. Please read the entire description of the connector before beginning fabrication! The RS-232 connector uses pins 1 (PC TXD), 5 (PC GND) and 6 (PC RXD) of the RJ-12 connector. This means that to properly wire a female DB9 to plug into the cRIO serial connector you need pin 1 of the RJ-12 to go to pin 3 of the DB9 connector. Next, pin 5 of the RJ-12 goes to pin 5 on the DB9 and pin 6 of the RJ-12 goes to

pin 2 of the DB9. This wires the DB9 as DCE (data communications equipment) and it uses a straight cable from the PC to the black jaguar. As it turns out, the cRIO DB9 is also wired as DTE (data terminal equipment) just as the PC is. So, the jaguar serial connector will plug straight into the cRIO without need for a null modem adapter. However, before you start crimping, you'll need to plan on where you're going to put the CAN bus termination resistor.

CAN bus is a true bus architecture. That is, it requires termination at both ends to work properly. The CAN specification calls for 120 ohm resistors at both ends of the bus. However, TI recommends 100 ohm resistors as being a bit more "robust" when the jaguars are used in robotics applications like the FRC. We're using 100 ohm resistors and they seem to work. Now, the resistor needs to go across the CAN H and CAN L signals (pins 3 & 4 of the RJ12). You can put the resistor into the same RJ-12 used for the serial port, or you can populate pins 3 & 4 with wires and put the resistor into the DB9 connector hood. Figure 5 shows the approach of mounting the terminator in the RJ-12 itself.

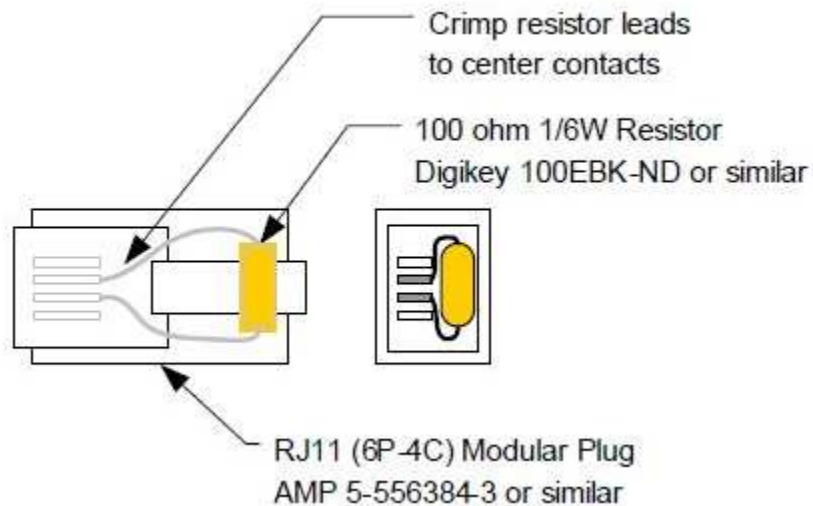


Figure 5. 100 ohm Terminator in RJ11/RJ12
(Source: TI)

My experience with the terminator in the RJ connector is that inserting and removing the RJ connector may cause the legs of the resistor to short out by having them touch each other. Personally, I'd recommend mounting the terminator inside the DB9 connector hood just to eliminate that possibility.

Once you've made up all of your cables, your jaguar daisy chain should look like Figure 6. Also, you can use fingernail polish or a dab of hotglue to hold the RJ-12s and terminators in place so things don't come loose if your robot should get bumped somewhere along the line ;-).

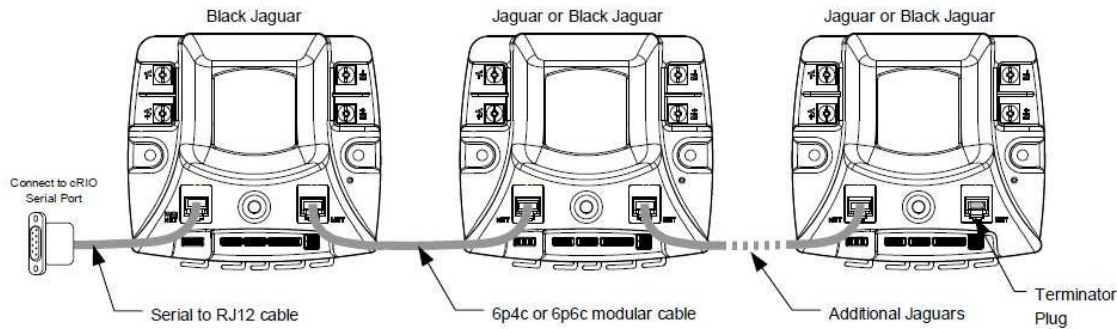


Figure 6. Jaguar Daisy Chain
(Source: TI)

Next, we move on to loading the firmware into the jaguars.

Burning Jaguar Firmware

Now that you've got your jaguar daisy chain put together, you've got to take it apart. By default, the jaguars ship with all of their CAN bus IDs set to ID 1. This won't work. So, we need to change the CAN IDs on each of the Jaguars to make them unique. Also, there is a special firmware load from TI that is specific to the FRC. This special firmware load includes a "trusted" CAN mode that isn't part of the normal product.

In the trusted mode, a watchdog timer on the cRIO will need to send out a keep-alive message every 100 ms or the jaguars will shut their motors down automatically. This is a nice safety feature to ensure that robots don't run amuck on the field. In order to load the new firmware, you'll need to use TI's BDC_COMM program and the firmware itself. Both of these are available for download at <http://www.luminarymicro.com/jaguar>.

For those of you using the 2CAN, that device can also load the firmware to the jaguars. See the 2CAN user's manual for instructions for how this is done.

Download the firmware and BDC_COMM application from the link listed above. Once you extract these .zip files, you'll have a BlackJaguar-87.bin and a Jaguar-87.bin file. These are the firmware loads for the black jaguar and the gray jaguar respectively. ***DO NOT MIX THESE FILES UP***. Burning the wrong firmware will likely brick your jaguar controller.

Next, with just the black jaguar connected and the terminator installed in the black jaguar, start the BDC_COMM applications (Windows only I'm afraid), select your COM port assignment and use the Status option to connect to the CAN bus as shown in Figure 7. If the jaguar is powered and the terminators are OK, you should see board ID 1 get selected and values changing in the display.

If you do not see any entries in the “Board ID” pull down, then something is wrong with your cabling/terminators.

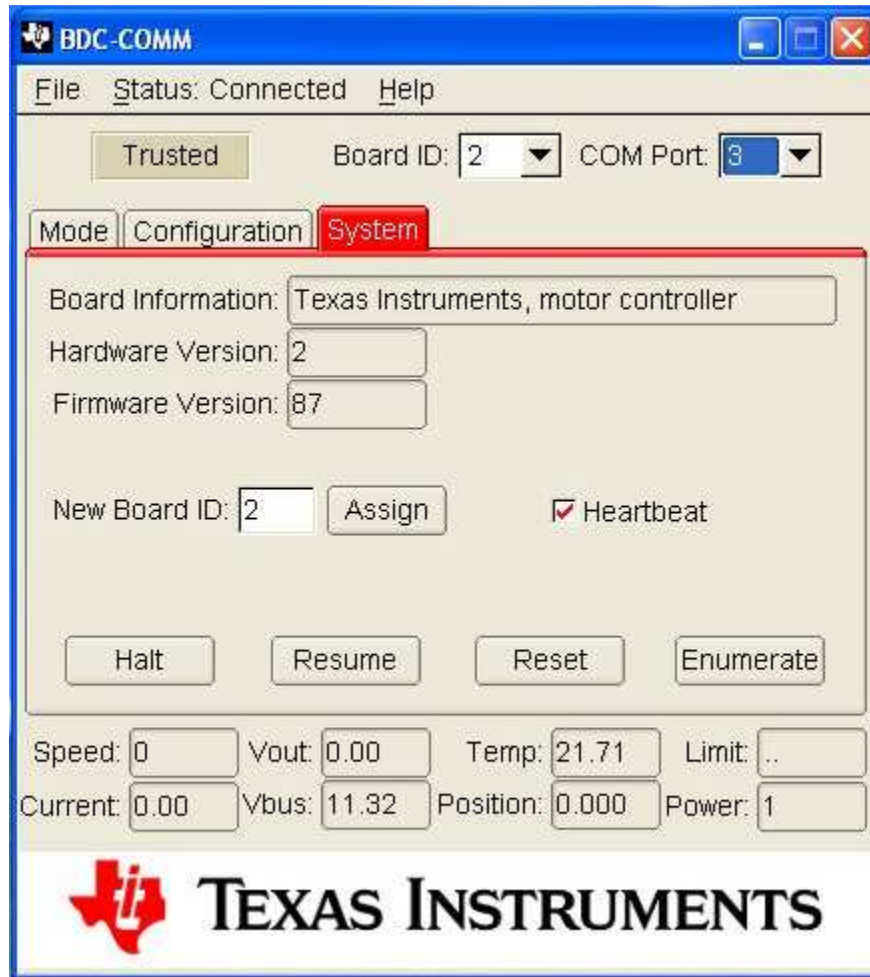


Figure 7. BDC_COMM After Firmware Update and ID Assignment

Updating the firmware requires that you use the BDC_COMM program's File->Update Firmware option and select the firmware for your unit as shown in Figure 8. Remember to burn the correct firmware to your jaguar. I am not responsible if you screw this step up! You have been warned...

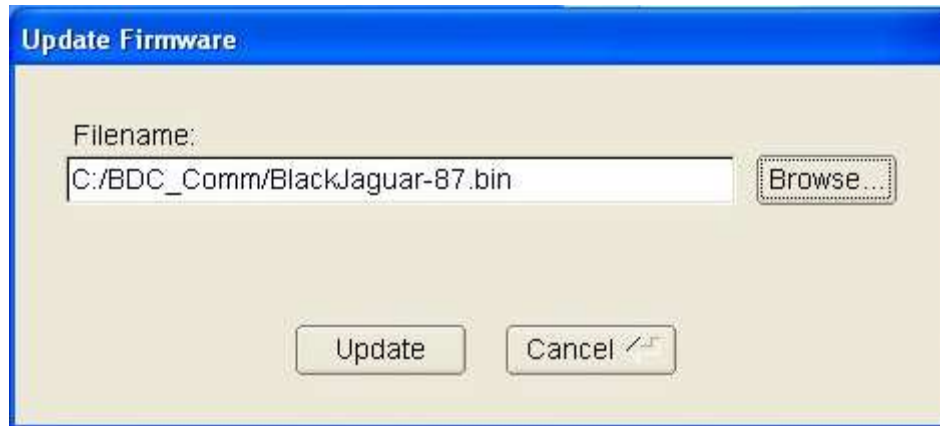


Figure 8. Update Jaguar Firmware (Black Jaguar only with this Firmware!)

The firmware update will take only a few seconds. Once complete, make sure that you reset your jaguar (power off and then back on) to get the new firmware in place. You should now see the label “Trusted” in the upper left hand corner of the BDC_COMM application.

Next, get a paper clip out and unbend it. You’ll need it to set the new CAN bus ID. In the dialog box labeled “New Board ID:” enter in a new ID > 1 and < 254 and press the “Assign” button. You’ll see the jaguar LED start blinking green and a 5-second countdown will start on the BDC_COMM screen. You have until it counts down to use the paperclip to press the “User” button on the top, right-hand side of the jaguar near the vents as seen in Figure 9. A failure to press the button in time will result in a jaguar without an address. If you miss the 5 second window, don’t panic. Simply assign the ID again and press the user button in time this next time.

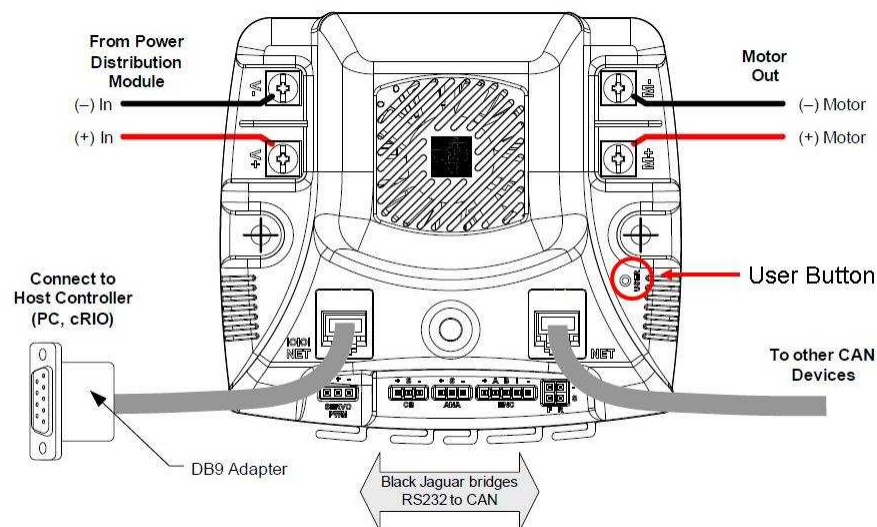


Figure 9. Location of User Button
(Source: TI)

Now that you've got the black jaguar set up. Move the terminator to the next jaguar, connect the CAN bus cable in the daisy chain and repeat the firmware download and ID assignment process making sure to use the correct firmware and a unique ID assignment. The new jaguar in the chain should be at ID 1. This is why you don't put more than one jaguar to be programmed into the chain at a time. Repeat the process until all of the jaguars have updated firmware and unique IDs.

Moving onto the cRIO

By default, the cRIO console serial port is enabled and running at 9600 baud. The console port is useful for debugging purposes and is the default STDOUT (e.g., printf and cout) for applications running on the cRIO platform. Normally, this port is configured as DTE, so a null modem RS-232 cable is required to interface between the cRIO and a PC.

Unfortunately, having the console port enabled for Wind River's VxWorks® (the operating system running on the cRIO) means that we can't use it for the CAN interface. So, in order to free up the console port while still having access to console messages, we need to take a couple more steps. First, we enable the netconsole facility and then we disable the cRIO's use of the serial port as a console.

If you are running at least V19 of the cRIO firmware (and you should be), then you already have the netconsole code on the cRIO. You simply need to enable it. This is done in the ni-rt.ini file located on the cRIO. Since there is no text editor on the cRIO, you'll need to FTP the ni-rt.ini file back to your development host and edit the file there. The first thing that you'll add to the file is the netconsole start up:

```
EarlyStartupLibraries=tsengine.out;NetConsole.out;
```

This will enable the netconsole code very early in the boot cycle just after the boot loader executes. So, you'll still be able to see almost all of the boot process as well as the loading of the various DLLs. But, don't copy the ni-rt.ini file back just yet because you're not through editing the file yet.

You'll also need to add the FRC_BlackJagBridgePlugin.out file to the "StartupDlls" entry of the ni-rt.ini file:

```
StartupDlls=debug.o;NiRioRpc.out;niorbs.out;NiViSrvr.out;ni  
vissvc.out;nivision.out;niserial.out;FRC_FPGA.out;FRC_Netwo  
rkCommunication.out;FRC_BlackJagBridgePlugin.out;FRC_UserPr  
ogram.out;
```


The black jaguar plugin must follow the FRC_NetworkCommunication plug-in in order for it to work. Now, you can save the file and FTP it back to the cRIO. You'll also have to FTP the FRC_BlackJagBridgePlugin.out file to the cRIO in the /c/ni-rt/system directory. The black jaguar plug-in can be found on the FirstForge site at:

<http://firstforge.wpi.edu/sf/frs/do/listReleases/projects.canjaguar/frs.blackjagbridgeplugin>

Once you've FTP'd the files back to the cRIO, you're ready for the next step. You'll need to disable the console output on the serial port by turning off the DIP switch label "CONSOLE OUTPUT" in the cRIO. This will free up the serial port for use by the jaguar code. There is an ominous note in the Getting Started with C guide says that there are reliability issues with rebooting if the Console port is disabled. I've not seen any such issues with the netconsole code enabled.

To complete this section of the CAN set up, you'll need to download and install the netconsole application (Windows only). It's available in the Team Update 4.0 section of the page at <http://first.wpi.edu/FRC/frccupdates.html>. Once the netconsole application is installed, reboot your cRIO. Assuming that you have all of the cabling and IP addresses set correctly, you should now see cRIO start up console output. BTW, the net console interface is bi-directional. You can actually talk to the cRIO and issue VxWorks® commands like "routeShow" and "i" via this interface. This makes an excellent debugging aid because you can dump memory (the "d <addr>" command), set breakpoints and perform a number of very low-level commands that are difficult to do from the Workbench facility.

Now for some CAN Code

Alright, assuming that you're still with me, you're almost ready to get your robot working with CAN bus. Back on the FirstForge site at http://firstforge.wpi.edu/sf/frs/do/listReleases/projects.canjaguar/frs.canjaguar_for_c you'll find the CANJaguar release .zip file. Download that and unzip it to one of your development host's directories. It will extract out one top-level directory called "CANJaguar for C++" and two subdirectories "CANJaguar" and "CANJaguarVoltageModeExample". I would suggest moving the CANJaguar directory into your Wind River header directory (c:\windriver\vxworks-6.3\target\h) because you'll need the headers frequently.

The CANJaguarVoltageModeExample code is a robot implementation that reads the voltages from the jaguars via the CAN bus while you steer the robot in tele-op mode. However, I've found a couple of issues with the code that complicate things a bit. First, trying to import the project in that directory can be tricky. Open Workbench and right-click in the Project Explorer tab. Next, select

import. In the import dialog, select General->Existing Projects into Workspace as shown in Figure 10.

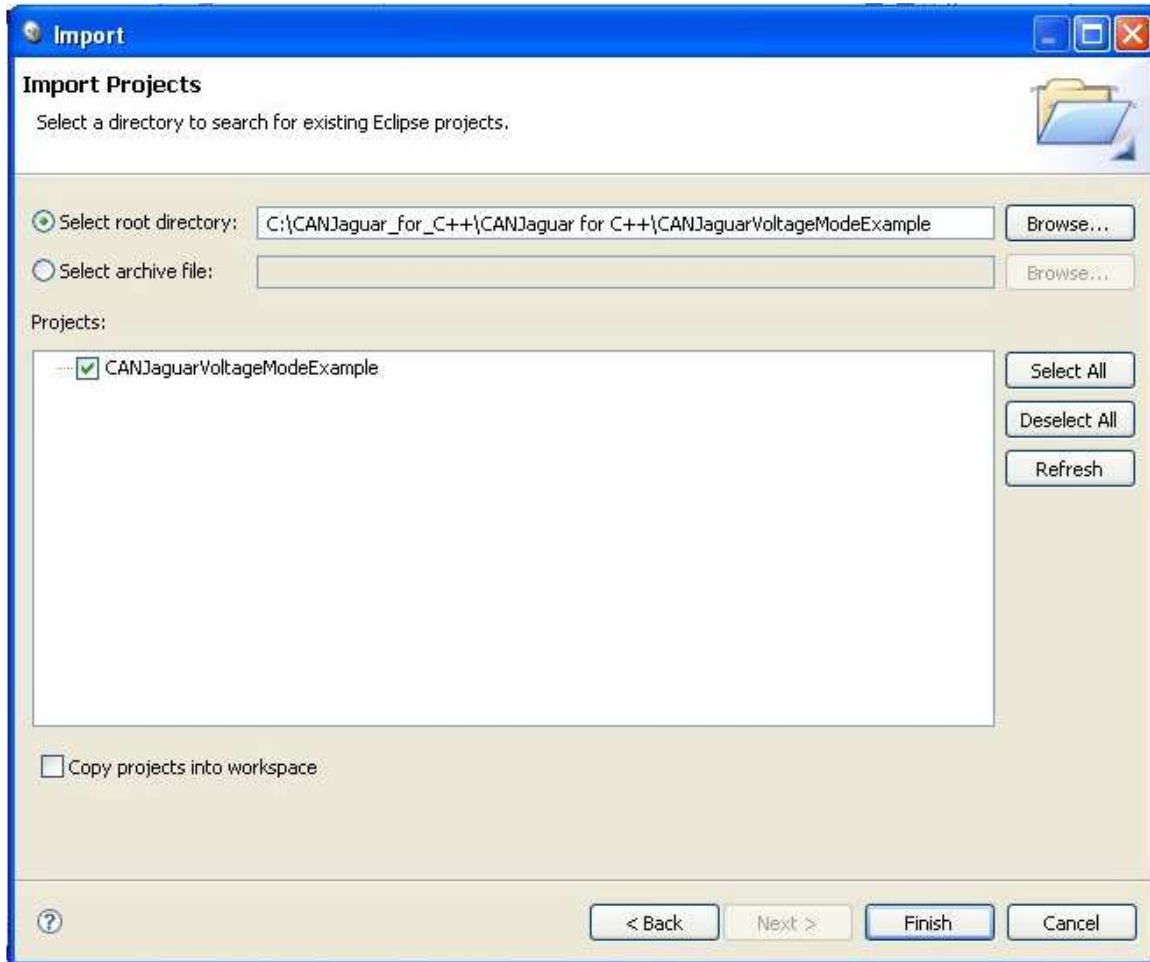


Figure 10. Import CAN Voltage Example Project

Once the project is imported, right-click the project and build it. This build should succeed. If it doesn't look at the Workbench build console output to try and determine where it went wrong.

OK, this looks good, but you're not done yet. The issue is that there is considerable code from the "CANJaguar" directory (CANJaguar.cpp) that your application will need. There are a couple of ways to get this code into your robot application. The first is to simply import the CANJaguar.cpp code into your Voltage Example project. To do this, right-click the Voltage Example project and select "import". Next, in the General section, select "File System" as shown in Figure 11.

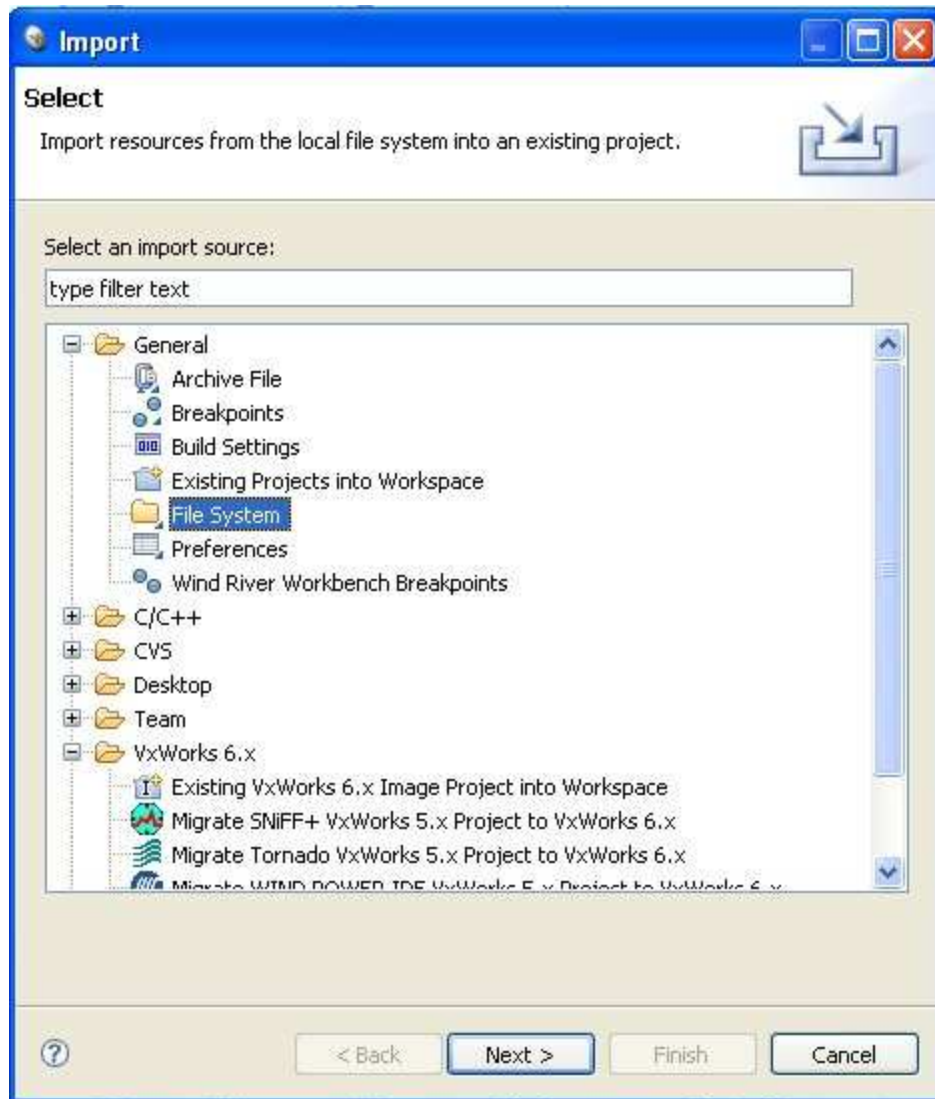


Figure 11. Import the CANJaguar.cpp code Step 1

Browse to the directory where the CANJaguar.cpp code is found and click “Next”. Select the CANJaguar.cpp file and click “finish” as shown in Figure 12.

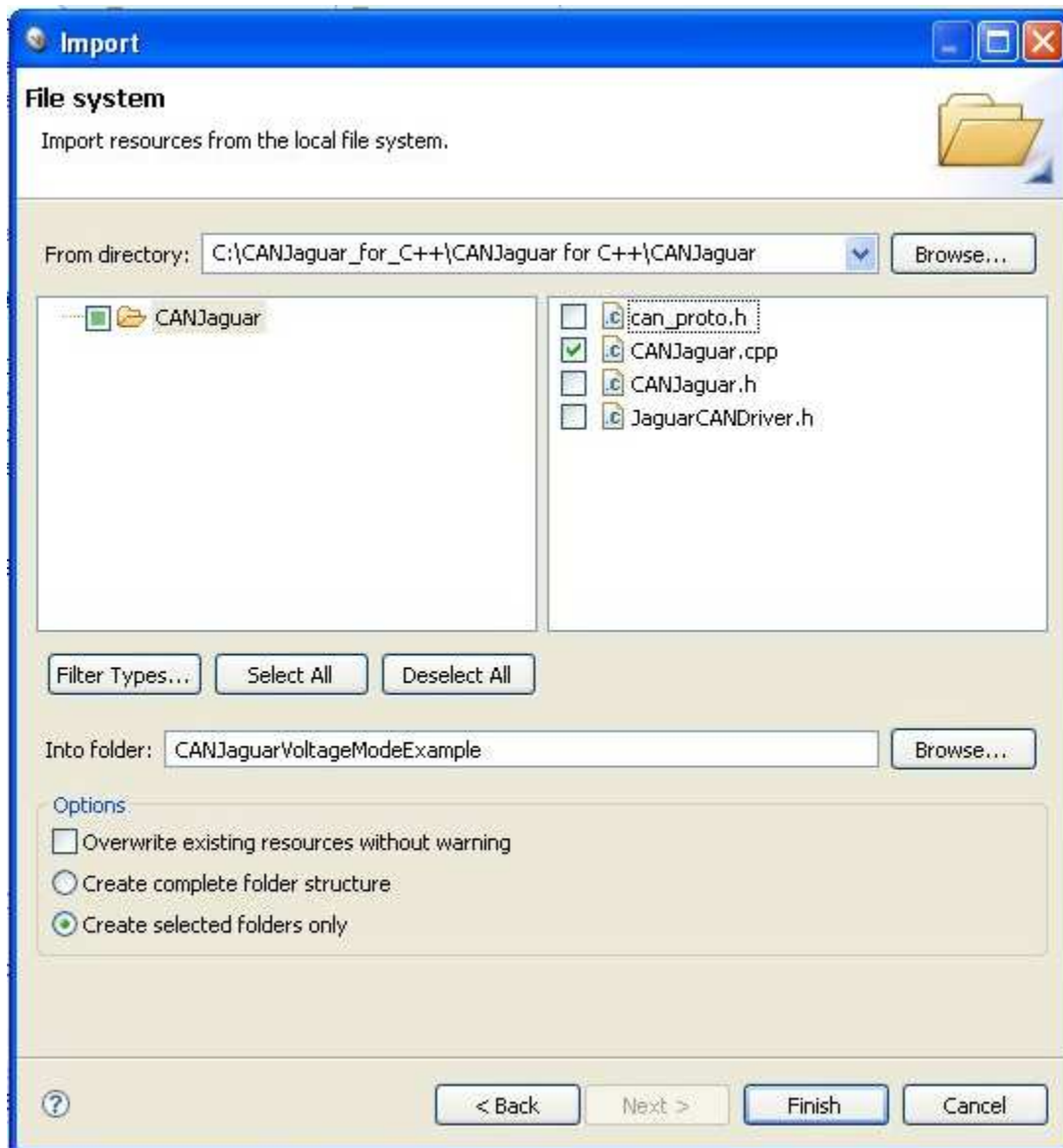
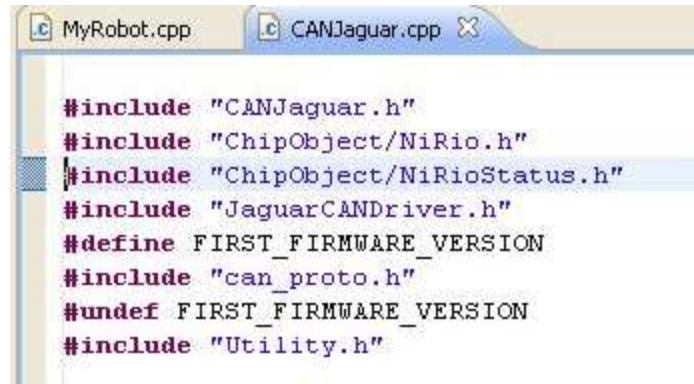


Figure 12. Import CANJaguar.cpp Step 2

Now, the CANJaguar.cpp code is part of your Voltage Example project and will be compiled and linked in when you build the project again. Unfortunately, there's a bug in this code. It's missing the "ChipObject/NiRio.h" header file. So, edit the CANJaguar.cpp source code and add:

```
#include "ChipObject/NiRio.h"
```

just above the `#include "ChipObject/NiRioStatus.h"` include directive and save the modified file. The CANJaguar.cpp file should now look like Figure 13.



```
#include "CANJaguar.h"
#include "ChipObject/NiRio.h"
#include "ChipObject/NiRioStatus.h"
#include "JaguarCANDriver.h"
#define FIRST_FIRMWARE_VERSION
#include "can_proto.h"
#undef FIRST_FIRMWARE_VERSION
#include "Utility.h"
```

Figure 13. Modified CANJaguar.cpp File

Next, we'll have to add the search path for the CANJaguar header files to the project. To do this, you'll right-click the project and select properties from the pop-up menu. Select "Build Properties" and the "Build Paths" tab. Now, add the header file path using the "Add" button. Assuming that you copied the CANJaguar directory into the VxWorks® header directory as was suggested earlier, your screen should look like Figure 14.

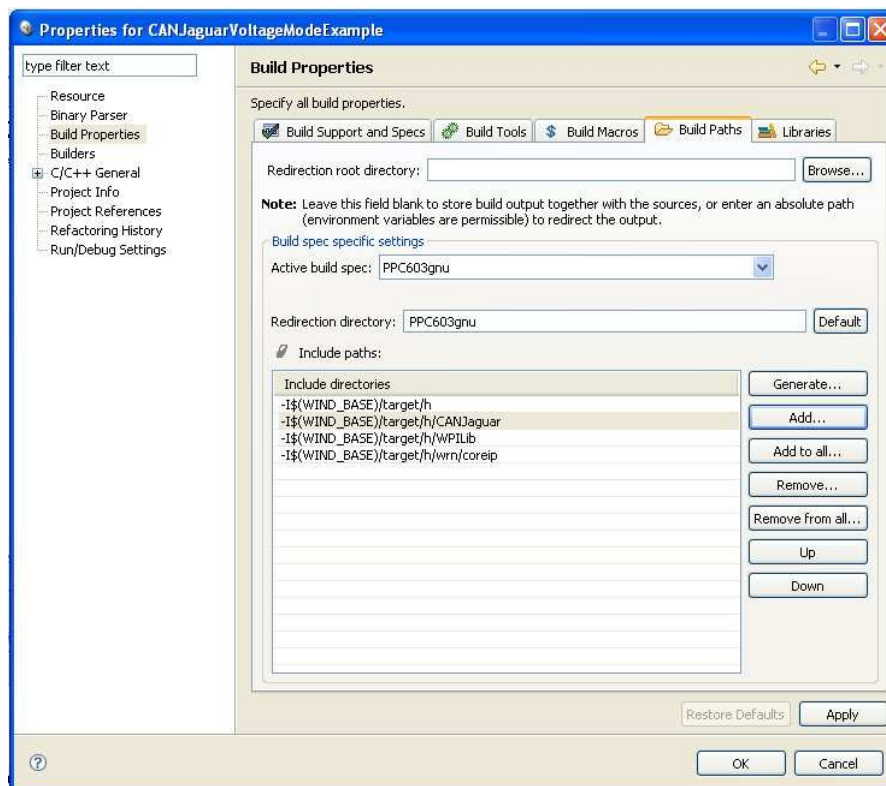


Figure 14. Setting up the Include Search Path

One last edit is needed to be made to the MyRobot.cpp file. In the instantiation of the "leftJag" and "rightJag" objects, you'll see that there are

numbers being passed. These are the CAN Bus IDs you assigned previously. Update the IDs with the appropriate IDs for your system and save the file. Now, you should be able to build the project successfully. At this point, the code is ready to deploy and run via the Driver Station. Watch the output in the netconsole and you should see voltages changing as you drive the robot.

Now, if you have more than one piece of code that interfaces with the CAN bus, the approach I used above to import the CANJaguar.cpp file into the Voltage Mode Example will result in multiple copies of the CANJaguar.cpp code being resident in the cRIO's memory. This is not good. So, your best bet is to make the CANJaguar.cpp code a DLL, FTP it to the cRIO and modify the ni-rt.ini file to automatically load the CANJaguar code at boot time like you did the FRC_BlackJagBridgePlugin.out earlier. This approach results in only one copy of the CANJaguar code in RAM and all subsequently loaded CAN code can use it.

Using CAN Bus

Whew! Is your brain full yet? At this point, I'd suggest taking a look at the Voltage Mode Example code and the CANJaguar.cpp and header files to see what else you can do with CAN bus. If you search around, there are some other references to various examples of CAN code on Chief Delphi that should help you out.

You get quite a bit of control and information via CAN bus that simply isn't available any other way. This can really help you control your robot when tied with other sources of information from the control system. And, there is a special "Sync" mode that allows you to preposition motor commands to each of the jaguars and then issue a single "activate" command so that all of the motors start moving at once. But, I'll leave that to you to figure out.

Summary

With all of this documented, we should all now have a better idea of what CAN bus is all about, how to set up the hardware and software and how to get a robot running using CAN bus. I focused on the TI Black Jaguar for this write up, but the basic mechanisms also apply to the 2CAN interface. I realize that this comes pretty late in the build season for most teams to be of much use. However, hopefully this document can serve as a reference for future projects or help as a debugging aid for teams that are still struggling with CAN bus control.

This is expected to be a living document. So, if you find errors or omissions or simply have a better way to do something, please let me know and I'll update the document. Good luck and I hope to see you on the field.