
FIRST Robotics Competition

WPILib and CTRE

Jul 24, 2020

Fundamentos de Programação e Elétrica

1	Primeiros passos	3
1.1	Introdução	3
1.2	Visão Geral dos Softwares de programação para FRC	4
1.3	Visão geral do Hardware do Sistema de Controle de FRC®	17
1.4	Offline Installation Preparation	35
1.5	Instalando LabVIEW para FRC (LabVIEW apenas)	36
1.6	Installing the FRC Game Tools	57
1.7	Como fazer o cabeamento de um robo para FRC	76
1.8	Formatando seu roboRIO	110
1.9	Programming your Radio	116
2	Getting Started with a Benchtop Robot	129
2.1	Creating your Benchtop Test Program (LabVIEW)	129
2.2	Running your Benchtop Test Program	134
3	FRC LabVIEW Programming	137
3.1	Creating Robot Programs	137
3.2	LabVIEW Resources	165
4	Atuadores	187
4.1	Visão geral dos atuadores	187
4.2	Operando cilindros pneumáticos	188
4.3	Usando controladores de motor no código	191
4.4	Controladores de velocidade PWM em profundidade	191
4.5	Usando WPILib para conduzir seu robô	192
4.6	Movimento repetitivo de baixa potência - servos de controle com WPILib	199
4.7	LEDs	199
5	Sensores	203
5.1	Sensor Overview - Software	203
5.2	Acelerômetros - Software	204
5.3	Gyroscopes - Software	206
5.4	Ultrasonics - Software	212
5.5	Contadores	215
5.6	Encoders - Software	222
5.7	Entradas analógicas - Software	231

5.8	Potenciômetros analógicos - Software	234
5.9	Digital Inputs - Software	235
6	CAN Devices	239
6.1	Using CAN Devices	239
6.2	Pneumatics Control Module	239
6.3	Power Distribution Panel	240
6.4	Third-Party CAN Devices	241
6.5	FRC CAN Device Specifications	243
7	Basic Programming	249
7.1	Git Version Control Introduction	249
8	Suporte	261
8.1	Outras documentações	261
8.2	Fóruns	261
8.3	Under Control 1156	261
8.4	Suporte via telefone da NI	262
8.5	CTRE	262
8.6	Reporte de Bugs	262
9	Phoenix Software Reference Manual	263
10	Primer: CTRE CAN Devices	265
11	Primer: What is Phoenix Software	267
11.1	What is Phoenix Tuner?	268
12	Prepare your workstation computer	271
12.1	Before Installing Phoenix...	271
12.2	What to Download (and why)	272
12.3	Workstation Installation	274
12.4	Post Installation Steps	279
13	FRC: Prepare NI roboRIO	283
13.1	Why prepare Robot Controller?	283
13.2	How to prepare Robot Controller	284
13.3	Verify the robot controller - Tuner	285
13.4	Verify the robot controller - LabVIEW	287
13.5	Verify the robot controller - Web page	289
13.6	Verify the robot controller - HTTP API	291
14	Initial Hardware Testing	295
15	Bring Up: CAN Bus	297
15.1	Understand the goal	297
15.2	Check your wiring	298
15.3	Power up and check LEDs	298
15.4	Open Phoenix Tuner	298
15.5	LEDs are red - now what?	299
15.6	Set Device IDs	301
15.7	Field upgrade devices	304
15.8	Pick device names (optional)	306
15.9	Self-test Snapshot	306
15.10	Driver Station Versions Page	308

16 Bring Up: PCM	311
16.1 Phoenix Tuner Self-test Snapshot	311
17 Bring Up: PDP	315
17.1 Getting sensor data	316
17.2 DriverStation Logs	317
17.3 2015 Kick off Kit PDPs	317
18 Bring Up: Talon FX/SRX and Victor SPX	319
18.1 Factory Default Motor Controller	319
18.2 Configuration	321
18.3 Test Drive with Tuner	322
18.4 Test Drive with Robot Controller	327
18.5 Open-Loop Features	330
18.6 Reading status signals	340
18.7 Limit Switches	341
18.8 Soft Limits	343
19 Troubleshooting and Frequently Asked Questions	345
19.1 Driver Station Messages	345
19.2 PCM	348
20 Driverstation	353
20.1 Imaging your Classmate (Veteran Image Download)	353
20.2 FRC Driver Station Powered by NI LabVIEW	375
20.3 Programming Radios for FMS Offseason	384
20.4 Troubleshooting Dashboard Connectivity	391
20.5 Driver Station Best Practices	393
20.6 Driver Station Log File Viewer	396
21 Networking Introduction	405
21.1 Networking Basics	405
21.2 IP Configurations	412
21.3 roboRIO Network Troubleshooting	414
21.4 Windows Firewall Configuration	416
21.5 Measuring Bandwidth Usage	421
21.6 OM5P-AC Radio Modification	429

Seja bem-vindo a documentação do sistema de controle da FRC! Aqui você irá encontrar um guia passo-a-passo para a correta configuração e programação dos componentes eletrônicos mais importantes utilizados na competição.

Atenção! Este guia foi adaptado para português da [Documentação oficial da FIRST](#) e da [Cross the Road Eletronics](#).

Primeiros passos

1.1 Introdução

Bem-vindo à documentação para os pacotes de software do Sistema de Controle e WPILib da *FIRST*® Robotics Competition. Esta página é a principal fonte de documentação quanto ao uso do Sistema de Controle da FRC® (incluindo fiação, configuração e software), assim como as bibliotecas e ferramentas da WPILib. Lembrando que esta é a versão traduzida pela equipe Under Control, para acessar a versão oficial em inglês, [clique aqui](#).

1.1.1 Novo na Programação?

Estas páginas cobrem os detalhes das bibliotecas WPILib e do Sistema de Controle da FRC e não descrevem o básico do uso das linguagens de programação suportadas. Se procura obter recursos para aprender as linguagens de programação suportadas, consulte as recomendações abaixo:

Note: Você pode continuar com esta seção de Introdução para obter um robô básico funcional sem o conhecimento da linguagem de programação. Para prosseguir, você precisará estar familiarizado com a linguagem de programação escolhida.

C++

- [LearnCPP](#)
- [Programming: Principles and Practice Using C++ 2nd Edition](#) is an introduction to C++ by the creator of the language himself (ISBN-10: 0321992784).
- [C++ Primer Plus 6th Edition](#) (ISBN-10: 0321776402).

Java

- [Code Academy](#)

- [Head First Java 2nd Edition](#) is a a very beginner friendly introduction to programming in Java (ISBN-10: 0596009208).

LabVIEW

- [National Instruments Learn LabVIEW](#)

1.1.2 Do zero á um robô

As páginas restantes nesta seção de Introdução foram projetadas para serem concluídas para ir de nada a um robô básico funcional. Os documentos guiarão você na instalação do software necessário, na fiação e configuração do hardware, e no carregamento de um exemplo básico que deve permitir a operação de seu robô. Quando você completar uma página, clique em **Next** para navegar para a próxima página e continuar com o processo. Quando estiver pronto, clique em **Next** para continuar para uma visão geral da WPILib em C++/Java ou clique no logo no canto superior esquerdo para voltar para página principal e explorar o conteúdo restante.

1.2 Visão Geral dos Softwares de programação para FRC

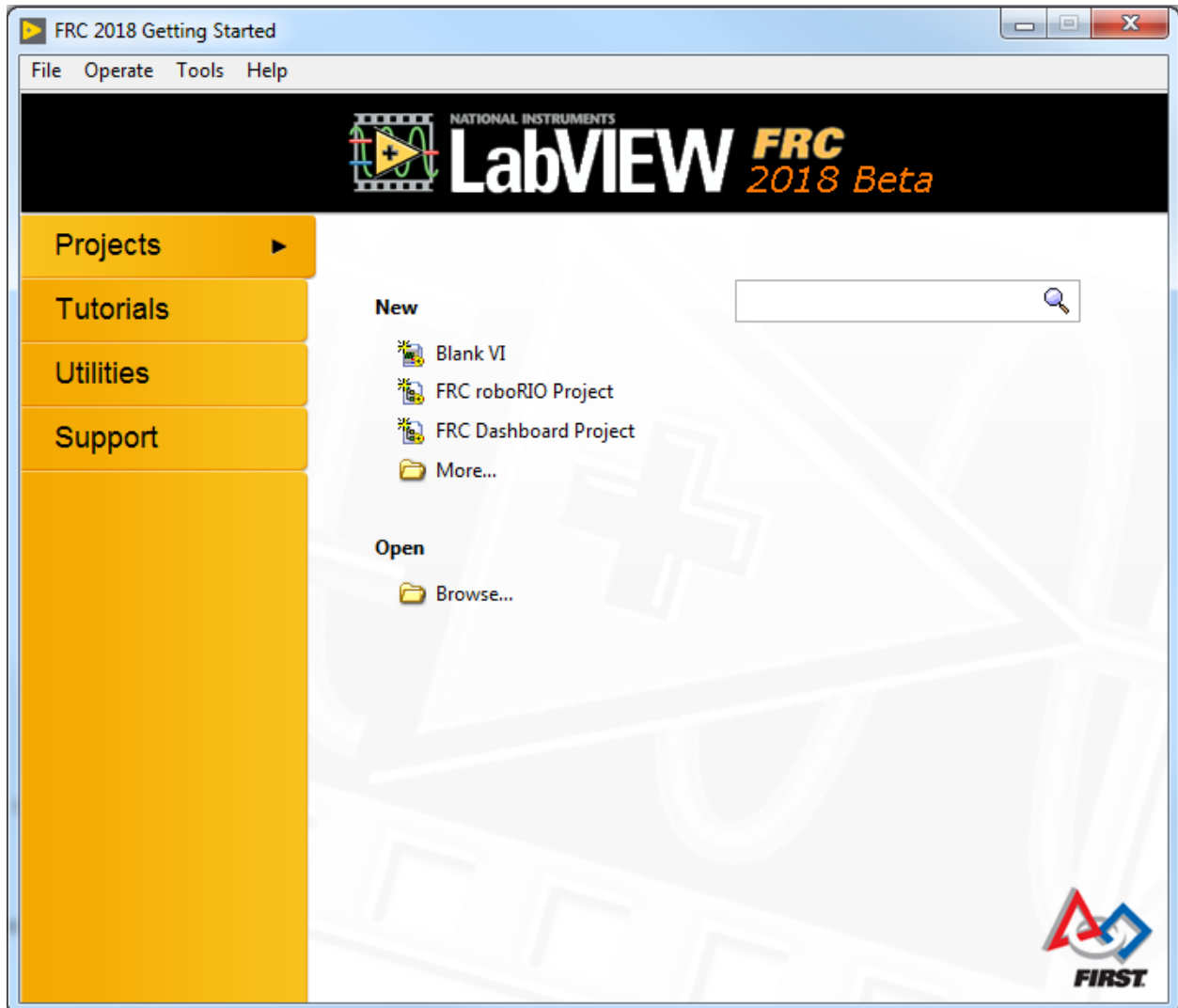
O sistema de controle FRC consiste em uma ampla variedade de componentes de software obrigatórios e opcionais. Estes elementos foram projetados para ajudá-lo no design, desenvolvimento, e depuração do código do robô, bem como, ajudar no controle operacional do robô e fornecer um feedback ao solucionar problemas. Para cada componente do software, este documento fornecerá uma breve visão geral de sua finalidade, um link para o download do pacote, se apropriado, e um link para a documentação adicional, quando disponível.

1.2.1 Compatibilidade com o Sistema Operacional

O sistema operacional principal com suporte para componentes FRC é o Windows. Todos os componentes de software FRC necessários foram testados no Windows 7, 8 e 10. O Windows XP não é suportado.

Dito isso, muitas das ferramentas para programação C++ / Java também são suportadas e testadas no macOS e Linux. As equipes que oprimem por programar em C++ / Java poderão desenvolver usando esses sistemas, usando um sistema Windows para operações como a Driver Station, a configuração do rádio e a formatação do roboRIO.

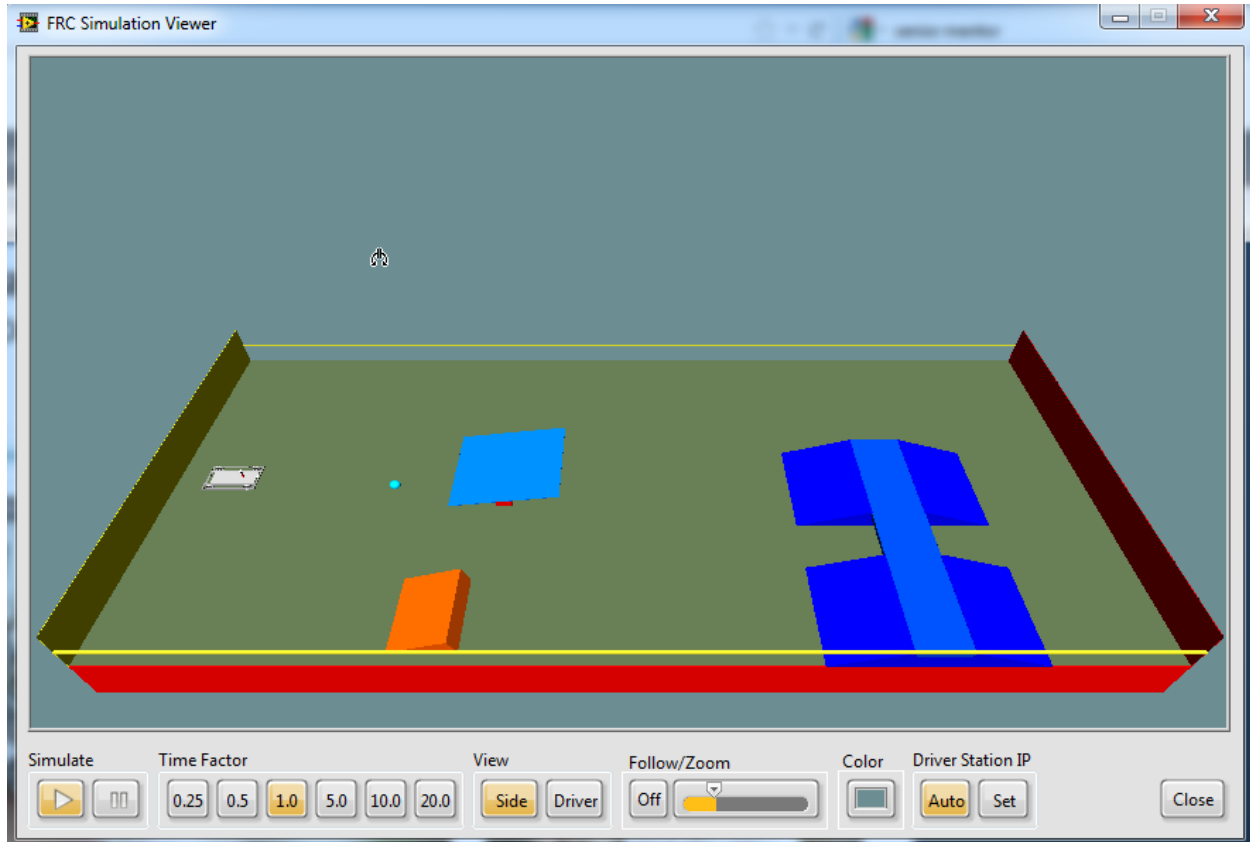
1.2.2 LabVIEW para FRC (Somente Windows)



LabVIEW para FRC, baseado no LabVIEW 2019 da National Instruments, é o ambiente de desenvolvimento do LabVIEW, uma das três linguagens oficialmente suportadas para a programação de um robô de FRC. O LabVIEW é uma linguagem gráfica orientada a fluxo de dados. Os programas no LabVIEW consistem em uma coleção de ícones, chamados VIs, conectados com fios que passam dados entre os VIs. O instalador do LabVIEW FRC é distribuído em um DVD encontrado no Kit de peças e também está disponível para download (consulte a página de instruções de instalação vinculada abaixo).

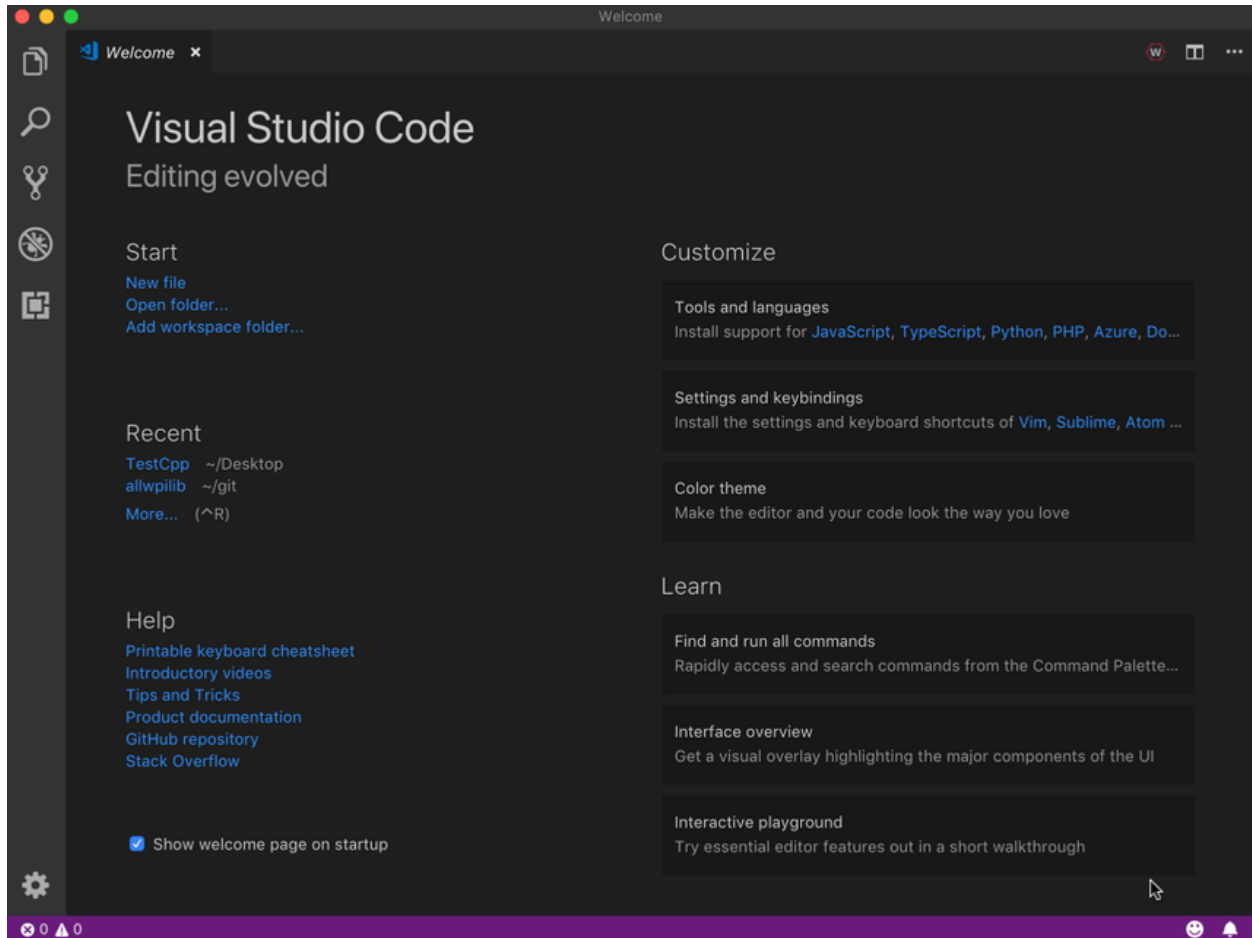
Instruções para instalar as bibliotecas para FRC (o pacote também inclui Driver Station e Utilitários) podem ser encontradas [aqui](#). Um guia para começar a usar o software LabVIEW FRC, incluindo instruções de instalação, pode ser encontrado [aqui](#).

Simulador de robô FRC (Somente Windows)



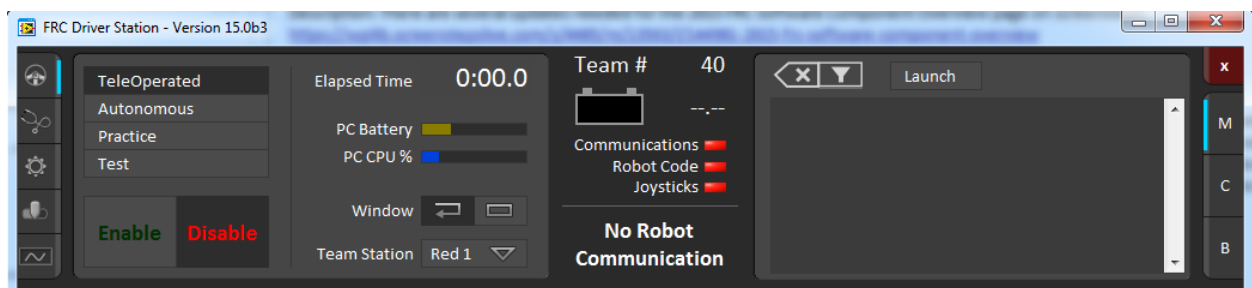
O simulador de robô FRC é um componente do ambiente de programação LabVIEW que permite operar um robô predefinido em um ambiente simulado para testar o código e/ou funções da Driver Station. Ele utiliza um projeto de código do LabVIEW como o código do robô e se comunica com a Driver Station para controle do robô e a Dashboard padrão para feedback do robô. O simulador de robô FRC é instalado com o pacote do LabVIEW para FRC. Informações sobre o uso do simulador de robô FRC podem ser encontradas abrindo o arquivo simulação do robô [Readme.html](#) no Explorador de Projetos do LabVIEW.

1.2.3 Visual Studio Code



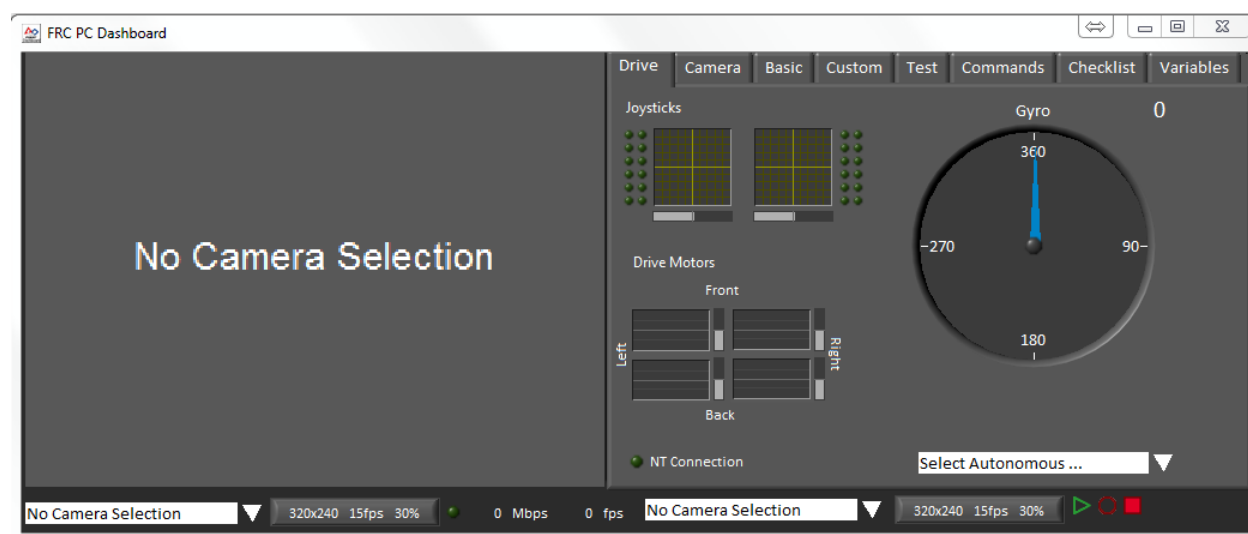
O Visual Studio Code é o ambiente de desenvolvimento suportado para C++ e Java, duas das três linguagens suportadas usadas para programar um robô FRC. Ambas linguagens de programação baseadas em texto orientadas a objetos. Um programa em C++ (para FRC) consiste em vários arquivos de cabeçalho (.h) e de implementação (.cpp), enquanto um programa em Java consiste em arquivos .java contidos em um ou mais pacotes. Um guia para começar a usar o C++ para FRC, incluindo a instalação e configuração do Visual Studio Code, pode ser encontrado [aqui](#).

1.2.4 Driver Station da FRC fornecida pela NI LabVIEW (Somente Windows)



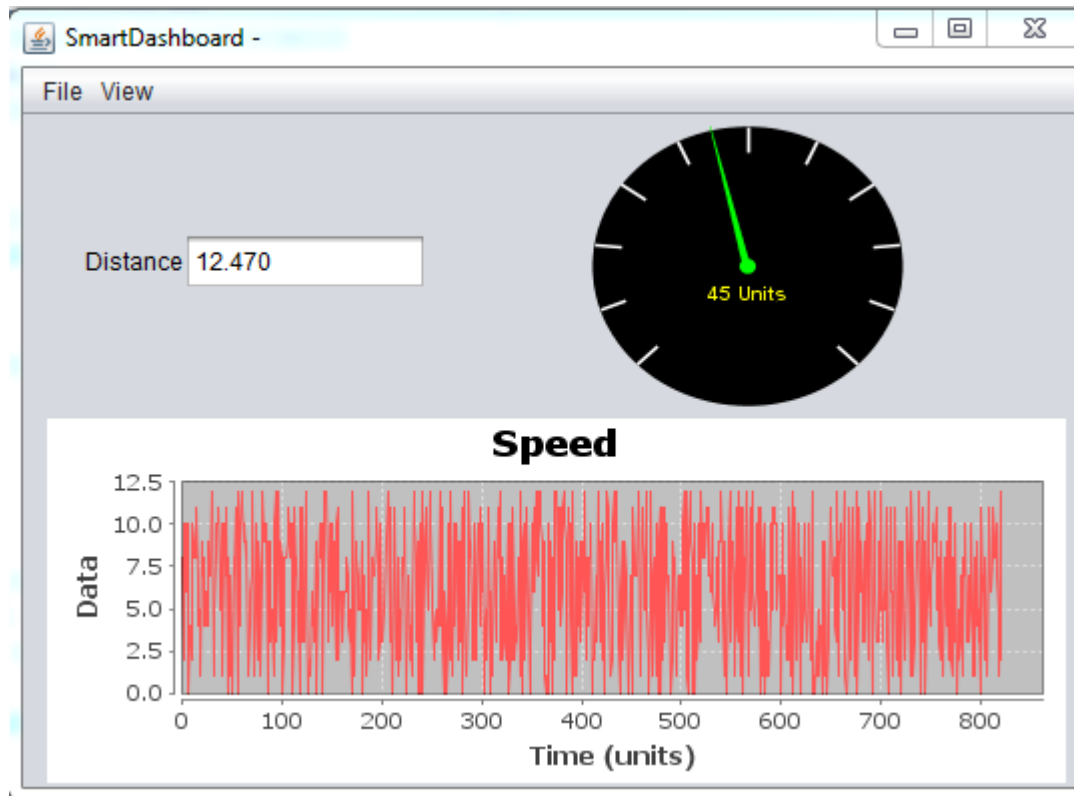
A Driver Station da FRC desenvolvido pelo NI LabVIEW é o único software permitido para ser usado com o objetivo de controlar o estado do robô durante a competição. Este software contém o código necessário para enviar dados ao seu robô a partir de uma variedade de dispositivos de entrada, como joysticks, gamepads e placas IO personalizáveis. Ele também contém várias ferramentas usadas para ajudar a solucionar problemas do robô, como indicadores de status e criação de arquivo de log. Instruções para instalar a Driver Station da FRC, fornecida pelo NI LabVIEW (incluída no FRC Game Tools, podem ser encontradas [aqui](#). Mais informações sobre a estação de driver FRC, fornecida pelo NI LabVIEW, podem ser encontradas [aqui](#).

1.2.5 Painel FRC LabVIEW (somente Windows)



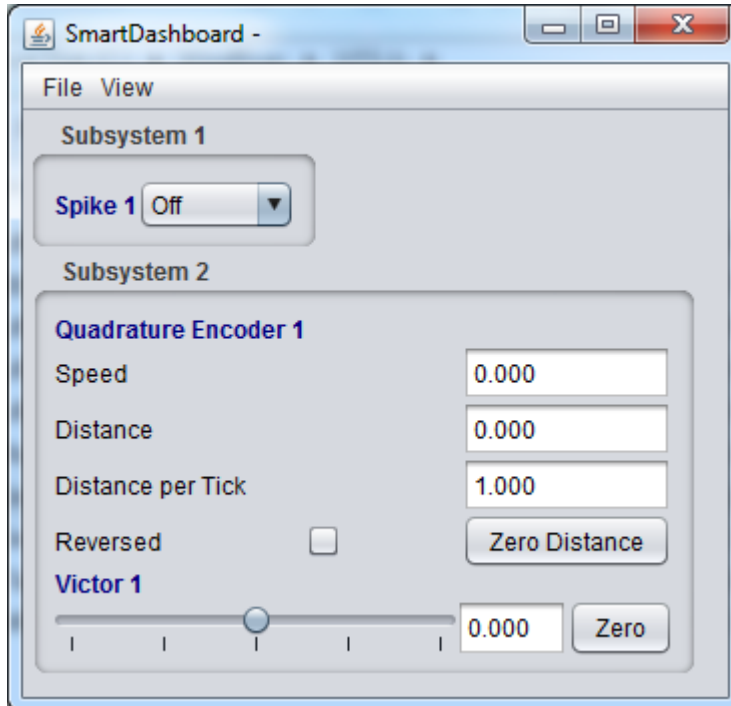
O FRC LabVIEW Dashboard é o programa padrão do painel instalado e iniciado automaticamente pela FRC Driver Station. O objetivo do painel é fornecer feedback sobre a operação do robô. O painel padrão da FRC serve como um exemplo dos tipos de feedback que as equipes podem desejar do robô. Ele inclui uma tela com guias que pode alternar entre a visualização de uma imagem de uma câmera no robô ou uma exibição das variáveis do NetworkTables, uma exibição de informações sobre os joysticks e os motores de acionamento, um indicador do IP do robô e da tensão da bateria e uma segunda aba exibição que pode alternar entre exemplos de indicadores e controles personalizados, uma guia de teste para uso com o Modo de Teste da Driver Station e uma guia lista de verificação que as equipes podem usar para inserir uma lista de verificação personalizada para concluir antes de cada partida. O painel padrão do FRC está incluído no FRC Game Tools. Instruções de instalação podem ser encontradas [aqui](#). Mais informações sobre o software do painel padrão FRC podem ser encontradas [aqui](#).

1.2.6 SmartDashboard



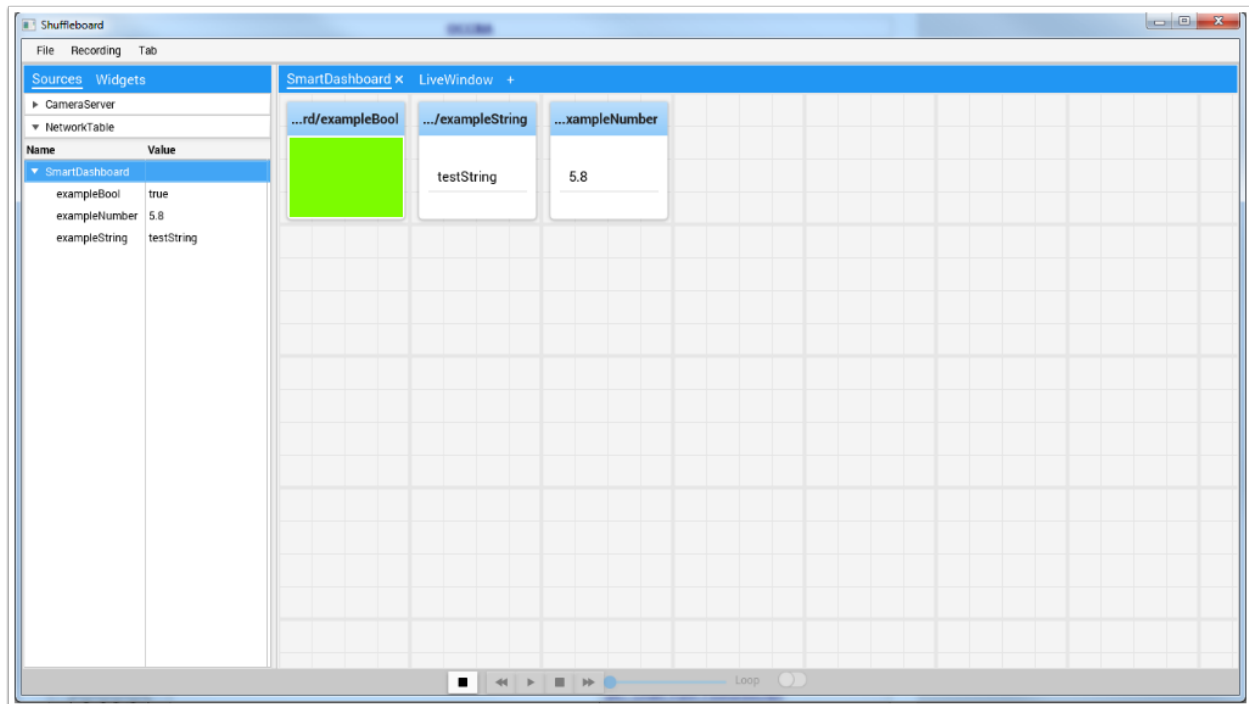
O SmartDashboard é um aplicativo de painel alternativo escrito no Java. O SmartDashboard cria automaticamente uma ferramenta para cada variável enviada do robô enviado usando a classe ou VIs do SmartDashboard. Essas ferramentas podem ser configuradas para vários tipos de exibição predefinidos ou os usuários podem criar extensões personalizadas em Java. As extensões de visão estão disponíveis para o SmartDashboard, que permite exibir imagens da câmera Axis no robô. O SmartDashboard está incluído nas atualizações de linguagem C++ e Java (ativadas clicando nos botões C++ ou Java, respectivamente, na guia Configuração do Driver Station). Documentação adicional no SmartDashboard pode ser encontrada aqui.

LiveWindow



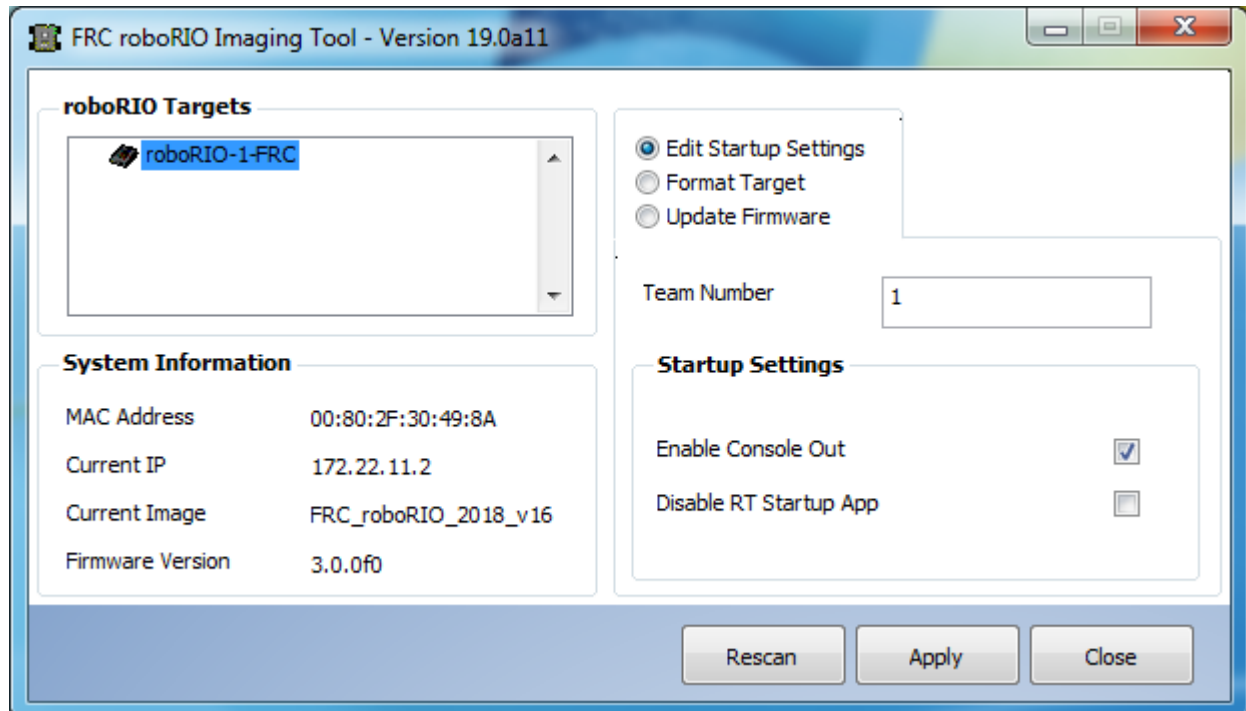
O LiveWindow é um modo do SmartDashboard, projetado para uso com o Modo de Teste da Driver Station. O LiveWindow permite que o usuário veja o feedback dos sensores no robô e nos atuadores de controle, independentemente do código do usuário. Mais informações sobre o LiveWindow podem ser encontradas [aqui](#).

1.2.7 Shuffleboard



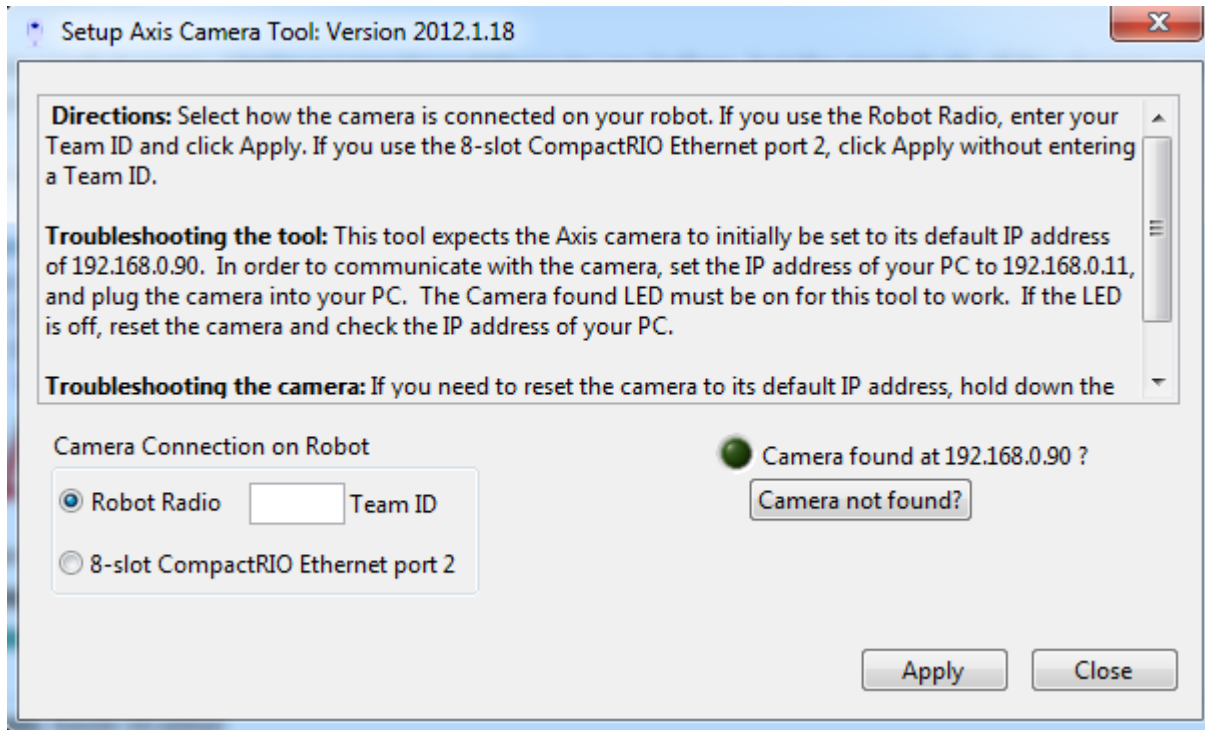
Shuffleboard é um aplicativo de painel alternativo escrito em Java. É preciso muitos dos conceitos do SmartDashboard, como adição automática de widgets e novos recursos, incluindo melhor controle de layout e funcionalidade de gravação / reprodução. O Shuffleboard contém todos os tipos básicos de widgets encontrados no SmartDashboard, além de vários novos destinados a tornar a visualização de componentes de robôs específicos ainda mais fácil. Possui total integração com o “cscore” do WPILib para exibir, gravar e reproduzir fluxos de câmera. O Shuffleboard está incluído nas atualizações de linguagem C++ e Java (habilitado selecionando Shuffleboard no tipo de painel na guia Configuração do Driver Station ou iniciando-o no menu da ferramenta de inicialização WPILib no Visual Studio Code). Documentação adicional no Shuffleboard pode ser encontrada aqui.

1.2.8 Ferramenta de imagem FRC roboRIO (Somente Windows)



A FRC roboRIO Imaging Tool é uma ferramenta de software usada para formatar e configurar um dispositivo roboRIO-FRC para uso no FRC. A ferramenta detecta qualquer dispositivo roboRIO na rede, relata o MAC atual, nome, IP e versão da imagem. A ferramenta permite que o usuário configure o número da equipe, defina opções como Saída do console e se um aplicativo é executado na inicialização e instale a imagem mais recente do software no dispositivo. A FRC roboRIO Imaging Tool é instalada como parte do FRC Game Tools. As instruções de instalação podem ser encontradas [aqui](#). Instruções adicionais sobre como criar imagens do seu roboRIO usando esta ferramenta podem ser encontradas [aqui](#).

1.2.9 Configuração da câmera Axis (Somente Windows)



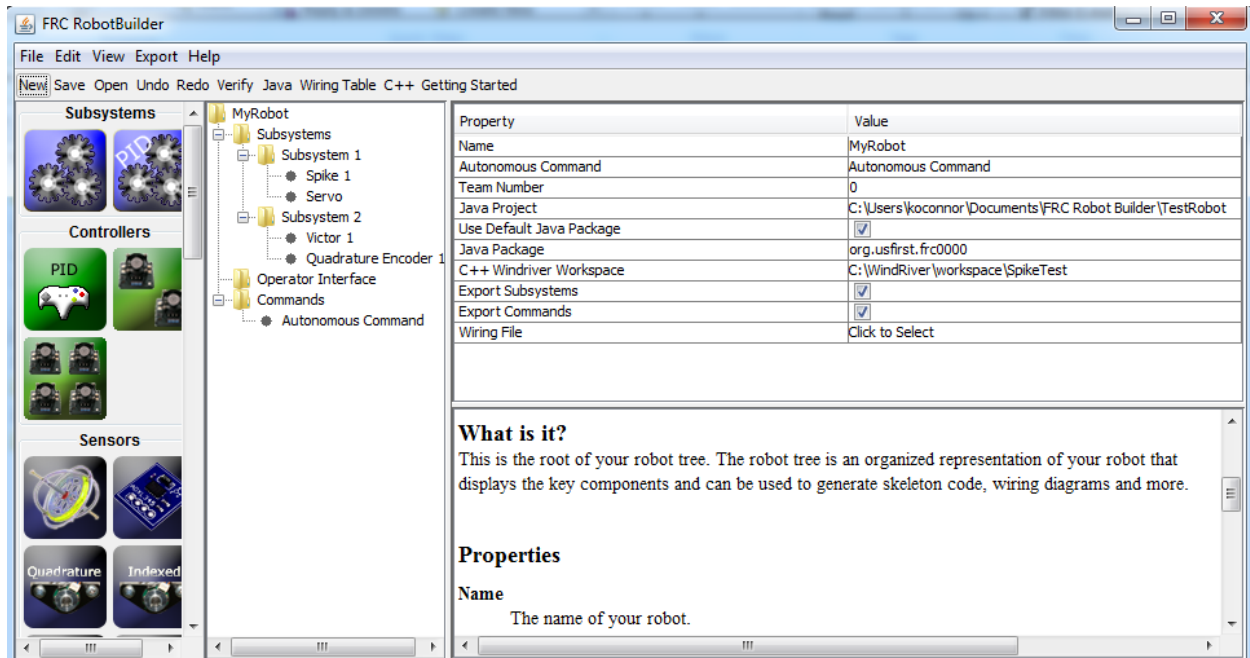
O utilitário Configuração da Câmera Axis é um programa do LabVIEW usado para configurar uma câmera Axis 206, M1011 ou M1013 para uso no robô. A ferramenta utiliza uma câmera de redefinição de fábrica conectada diretamente ao computador e configura o IP, nome de usuário e senha, acesso anônimo e taxa de quadros padrão e compressão (para uso com o SmartDashboard ou outros métodos de acesso). A ferramenta Setup Axis Camera é instalada como parte do FRC Game Tools. As instruções de instalação podem ser encontradas [aqui](#). As instruções para usar a ferramenta para configurar a câmera estão localizadas aqui.

1.2.10 Visualizador de Registros da FRC Driver Station (Somente Windows)



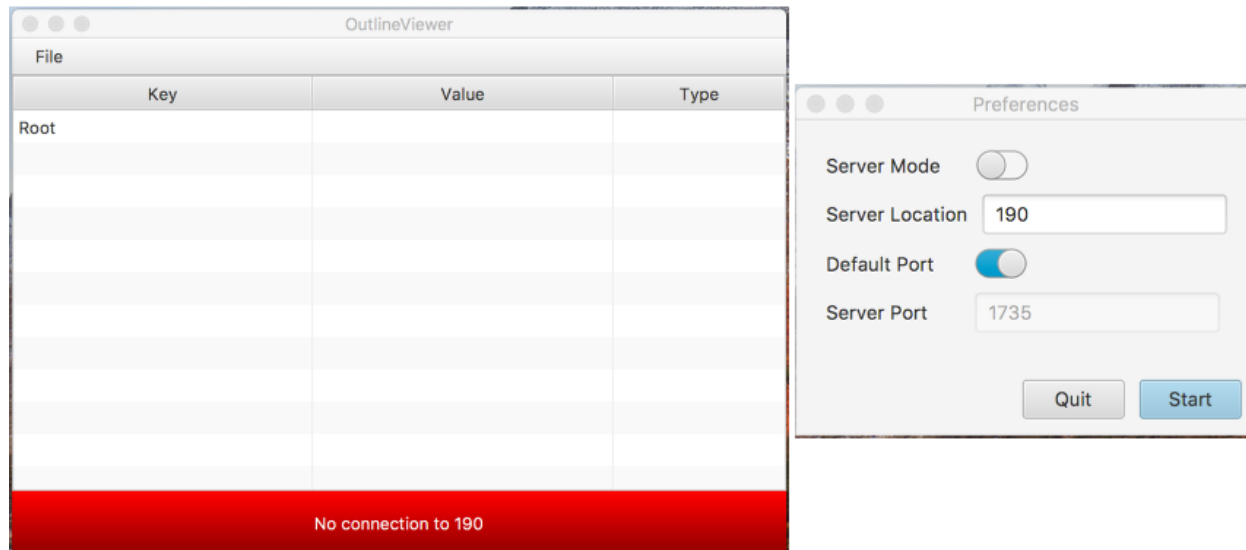
O Visualizador de Registros da FRC Driver Station é um programa LabVIEW usado para visualizar registros criados pelo FRC Driver Station. Esses registros contêm informações como voltagem da bateria, tempo de disparo, CPU% e modo do robô, além de eventos como remoção do joystick. O visualizador de registro da FRC Driver Station está incluído no FRC Game Tools. As instruções de instalação podem ser encontradas [aqui](#). Mais informações sobre o visualizador de registros da estação de driver FRC e a compreensão dos registros podem ser encontradas [aqui](#).

1.2.11 RobotBuilder



O RobotBuilder é uma ferramenta projetada para ajudar na configuração e estruturação de um projeto de robô baseado em comando para C++ ou Java. O RobotBuilder permite que você insira os vários componentes dos subsistemas do robô e da interface do operador e defina quais são seus comandos em uma estrutura de árvore gráfica. O RobotBuilder verificará que você não possui conflitos de alocação de porta e pode gerar uma tabela de ligações indicando o que está conectado a cada porta, bem como o código C++ ou Java. O código criado gera os arquivos apropriados, constrói os objetos apropriados e adiciona o código do LiveWindow para cada sensor e atuador, mas não grava nenhum dos métodos reais de subsistema ou comando. O usuário deve escrever o código apropriado para esses métodos para o robô funcionar. Mais informações sobre o RobotBuilder podem ser encontradas aqui. Mais informações sobre a arquitetura de programação baseada em comando podem ser encontradas aqui.

1.2.12 OutlineViewer

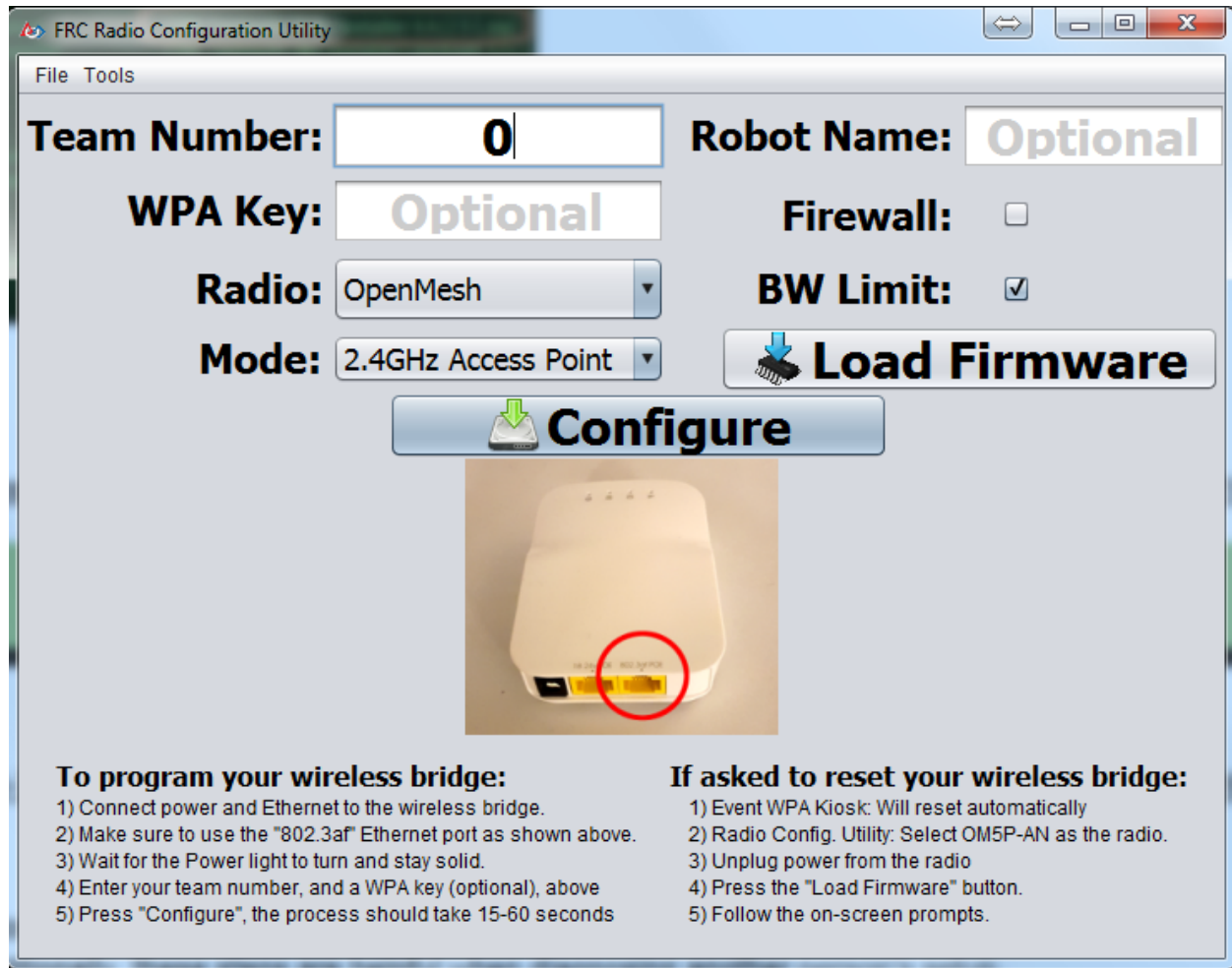


O OutlineViewer é um utilitário usado para exibir, modificar e adicionar ao conteúdo das Tabelas de Rede para fins de depuração. Ele exibe todos os pares de valores de chave atualmente nas tabelas de rede e pode ser usado para modificar o valor das chaves existentes ou adicionar novas chaves para a tabela. O OutlineViewer está incluído nas atualizações de linguagem C++ e Java (encontradas em `\tools\wpilib`). As equipes talvez precisarão instalar o Java Runtime Environment para usar o OutlineViewer em computadores não configurados para programação Java.

Para conectar-se ao seu robô, abra OutlineViewer e defina a “localização do servidor” como o seu número da sua equipe. Depois de clicar iniciar, OutlineViewer se conectará.

As equipes do LabVIEW podem usar a guia Variáveis do LabVIEW Dashboard para realizar essa funcionalidade.

1.2.13 Utilitário de configuração de rádio FRC (Somente Windows)



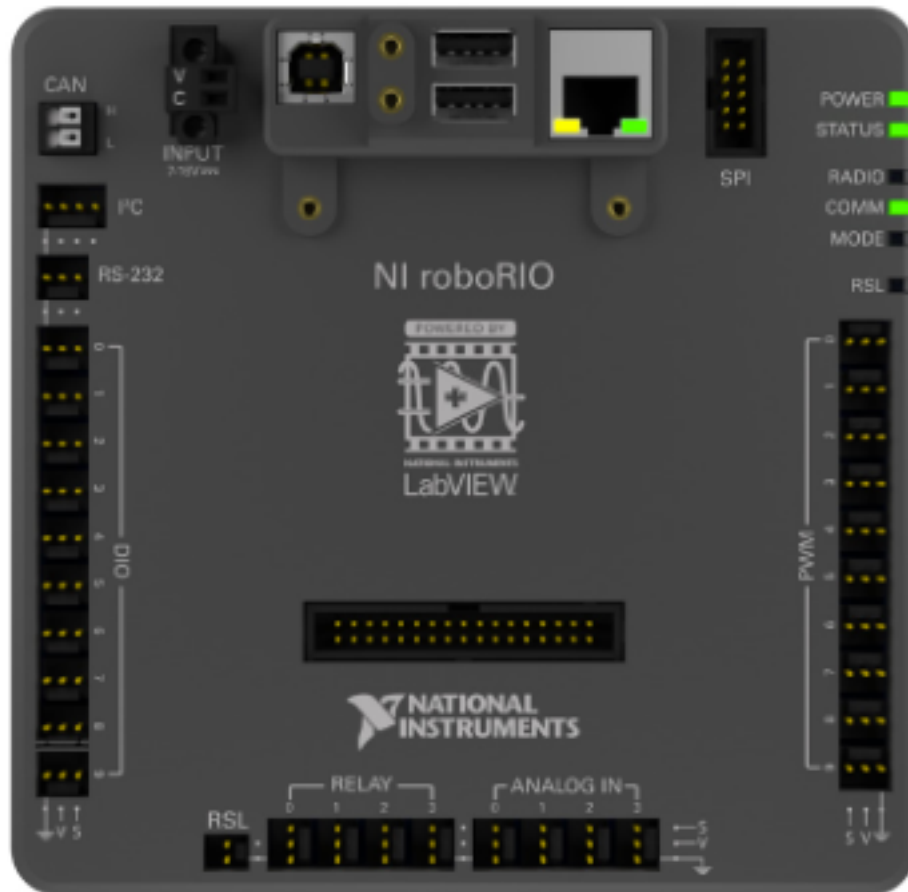
O FRC Radio Configuration Utility é uma ferramenta usada para configurar o rádio OpenMesh OM5P-AN ou OM5P-AC para uso prático em casa. Essa ferramenta define as configurações de IP e de configurações de rede para a conexão de rede adequada, bem como as configurações de QOS necessárias para imitar a experiência de limitação de largura de banda e priorização de pacotes no campo de jogo do FRC. O utilitário de configuração de rádio FRC é instalado por um instalador autônomo; instruções sobre a instalação e o uso do utilitário de configuração de rádio FRC para configurar seu rádio podem ser encontradas [aqui](#).

1.3 Visão geral do Hardware do Sistema de Controle de FRC®

O objetivo desse documento é fornecer uma breve visão geral dos componentes de hardware que compõem o Sistema de Controle de FRC®. Cada componente vai conter uma breve descrição da função do componente, uma breve lista de conexões críticas, e um link para mais documentação se disponível.

Note: Para instruções/diagramas de fiação completos, acesse o documento [Fiação do Sis-](#)

1.3.1 National Instruments roboRIO



O NI-roboRIO é o principal controlador de robô usado para FRC. O roboRIO inclui um processador dual-core ARM Cortex™-A9 e FPGA que executa os elementos confiáveis para controle e segurança, bem como o código gerado pela equipe. O Controlador integrado I/O inclui uma variedade de protocolos de comunicação (Ethernet, USB, CAN, SPI, I2C, e serial) como PWM, servo, digital I/O, e canais analógicos I/O usados para conectar os periféricos do robô para detecção e controle. O roboRIO deve conectar-se à porta de 12V no power distribution panel. A comunicação com fio está disponível via USB ou Ethernet. Informações detalhadas sobre o roboRIO podem ser encontradas no [Manual do Usuário do roboRIO](#).

1.3.2 Power Distribution Panel



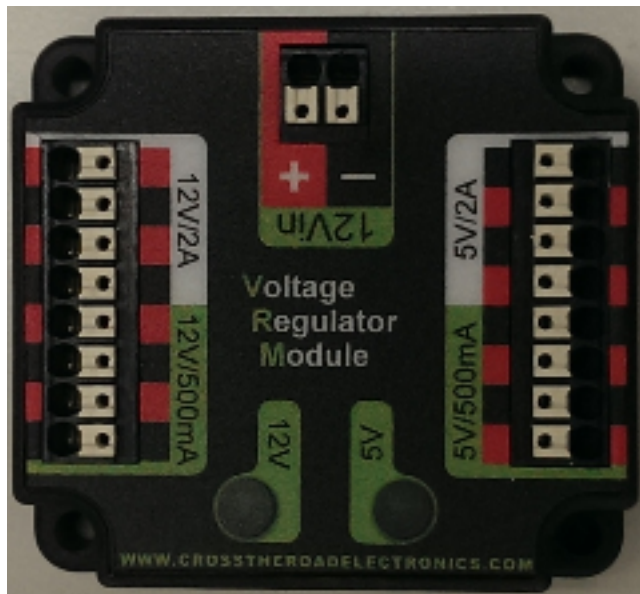
A Power Distribution Panel (PDP) foi projetada para distribuir energia de uma bateria 12VDC para vários componentes do robô por meio de disjuntores com redefinição automática e um pequeno número de conexões com funções especiais. A PDP oferece 8 pares de saída classificados para corrente contínua de 40A e 8 pares classificados para corrente contínua de 30A. A PDP oferece conectores dedicados de 12V para o roboRIO, bem como conectores para o Voltage Regulator Module (VRM) e para o Pneumatics Control Module (PCM). Ela também inclui uma interface CAN para registrar a corrente, temperatura e tensão da bateria. Para informações mais detalhadas, consulte o [PDP User Manual](#).

1.3.3 Pneumatics Control Module



A PCM é um dispositivo que contém todas as entradas e saídas necessárias para operar solenóides pneumáticas de 12V ou 24V e o compressor. A PCM é ativada/desativada pelo roboRIO através da interface CAN. A PCM contém uma entrada para o sensor de pressão e controlará o compressor automaticamente quando o robô estiver ativado e uma solenóide tiver sido criado no código. O dispositivo também coleta informações de diagnóstico, como o estado das solenóides, o estado do sensor de pressão, e o estado do compressor. O módulo inclui LEDs de diagnóstico para os canais de solenóide e CAN individualmente. Para mais informações, consulte o [Manual do Usuário da PCM](#).

1.3.4 Voltage Regulator Module



O VRM é um módulo independente que é alimentado por 12 volts. O dispositivo está conectado a um conector dedicado a ele na PDP. O módulo possui várias saídas reguladas de 12V e 5V. O objetivo do VRM é fornecer energia regulada para o rádio do robô, circuitos personalizados e câmeras de visão IP. Os dois pares de conectores associados a cada etiqueta têm uma classificação combinada do que a etiqueta indica (por exemplo, 5V / 500mA total para ambos os pares, não para cada par). O limite de 12V / 2A é uma classificação de pico, a fonte não deve ser carregada com mais de 1,5A de corrente contínua. Para mais informações, consulte o [Manual do Usuário do VRM](#).

1.3.5 Controladores de Motor

Há uma variedade de controladores de motor diferentes que funcionam com o sistema de controle FRC e são aprovados para uso. Esses dispositivos são usados para fornecer controle de tensão variável dos Brushed DC Motors usados na FRC. Eles estão listados aqui em ordem alfabética.

Controlador SPARK Motor



O Controlador SPARK Motor da REV Robotics é um controlador Speed Motor para uso na FRC. O SPARK é controlado utilizando a interface PWM. Os Limit switches podem ser conectados diretamente ao SPARK para limitar o deslocamento do motor em uma ou ambas as direções. O LED de status RGB exibe o estado atual do dispositivo, incluindo se o dispositivo está atualmente em Brake mode ou Coast mode. Para mais informações, acesse o [Página do produto REV Robotics SPARK](#)

SPARK MAX Motor Controller



O Controlador SPARK MAX Motor da REV Robotics é um controlador Speed Motor para uso na FRC. O SPARK MAX é capaz de controlar tanto os tradicionais Brushed DC Motors comumente usados na FRC ou o novo Brushless REV Robotics NEO Brushless Motor. O SPARK MAX pode ser controlado por meio da PWM, CAN ou USB (para configuração/testagem apenas). O controlador possui uma porta de dados para entrada do sensor e é capaz de assumir modos de closed loop control quando controlado por meio CAN ou USB. Para mais informações, acesse [Página do Produto REV Robotics SPARK MAX](#).

Controlador Talon Motor



O Controlador Talon Motor pela Cross the Road Electronics é um controlador Speed Motor para uso na FRC. O Talon é controlado por meio de interface PWM . O Talon deve ser conectado a uma saída PWM do roboRIO e alimentado pela Power Distribution Panel. Para mais informações, acesse o [Manual do Usuário do Talon](#).

Talon SRX



O Controlador Talon SRX motor é um “controlador de motor inteligente” habilitado para CAN da Cross The Road Electronics/VEX Robotics. O Talon SRX possui um compartimento de metal eletricamente isolado para dissipação de calor, tornando opcional o uso de um ventilador. O Talon SRX pode ser controlado por meio CAN bus ou por interface PWM. Ao usar o controle CAN bus, esse dispositivo pode receber entradas de limit switches e potentiometers, encoders, ou sensores similares para executar um controle avançado como limitar ou PID(F) closed loop control no dispositivo. Para mais informações acesse o [Manual do Usuário do Talon SRX](#).

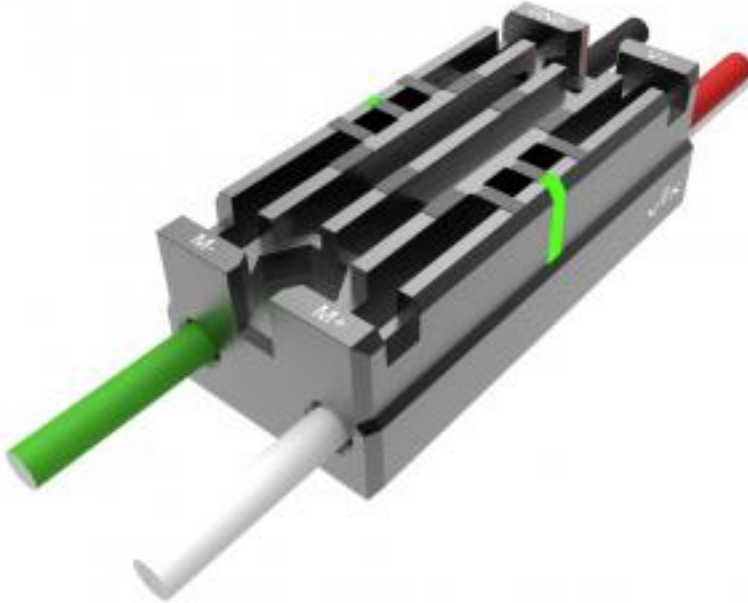
Note: CAN Talon SRX foi removido do WPILib. Acesse [blog](#) para mais informações e encontre o Instalador CTRE Toolsuite [aqui](#).

Controlador Victor 888 Motor / Controlador Victor 884 Motor



O Controlador Victor 888 Motor da VEX Robotics é um controlador Speed Motor para uso na FRC. O Victor 888 substitui o Victor 884, que também pode ser utilizado na FRC. O Victor é controlado por interface PWM. O Victor deve ser conectado a uma saída PWM output do roboRIO e alimentado pela Power Distribution Panel. Para mais informações, acesse o [Manual do Usuário do Victor 884](#) e [Manual do Usuário do Victor 888](#).

Victor SP



O Victor SP motor é um controlador PWM da Cross The Road Electronics/VEX Robotics. O Victor SP possui uma carcaça de metal eletricamente isolada para dissipação de calor, tornando opcional o uso do ventilador. O dispositivo é selado para impedir que detritos entrem no controlador. O controlador é aproximadamente metade do tamanho dos modelos anteriores.

Victor SPX



O Victor SPX motor é um controlador de motor por meio CAN ou PWM da Cross The Road Electronics/VEX Robotics. O dispositivo é conectado para permitir a fácil conexão com os conectores roboRIO, PWM ou CAN bus chain. Quando controlado por meio CAN bus, o dispositivo tem um número de recursos de closed loop também presente no Talon SRX. O dispositivo é selado para impedir que detritos entrem no controlador. Para mais informações, acesse a [Victor SPX Webpage](#).

Note: O controlador Victor SPX CAN não é suportado pelo WPILib. Acesse [this blog](#) para mais informações e encontre o instalador CTRE Toolsuite [aqui](#).

1.3.6 Spike H-Bridge Relay



O Spike H-Bridge Relay da VEX Robotics é um dispositivo usado para controlar a energia de motores ou outros componentes eletrônicos de robôs personalizados. Quando conectado a um motor, o Spike oferece On/Off control nas direções direta e reversa. As saídas do Spike outputs são independentemente controladas portanto também pode ser usado para fornecer energia para até 2 circuitos eletrônicos personalizados. O Spike H-Bridge Relay deve ser conectado a uma saída relay do roboRIO e alimentado pela Power Distribution Panel. Para mais informação [Guia do Uusário do Spike](#).

1.3.7 Servo Power Module



O Servo Power Module da Rev Robotics é capaz de expandir a energia disponível para os servos além do que a fonte de alimentação integrada do roboRIO é capaz. O Servo Power Module fornece até 90W de potência de 6V em 6 canais. Todos os sinais de controle são transmitidos diretamente do roboRIO. Para mais informações, acesse a [Servo Power Module Webpage](#).

1.3.8 Microsoft Lifecam HD3000



A Microsoft Lifecam HD3000 é uma webcam USB que pode ser conectada diretamente ao roboRIO. A câmera é capaz de capturar vídeo de até 1280x720 a 30 FPS. Para mais informações sobre a câmera, consulte a [Página do produto Microsoft](#). Para obter mais informações sobre o uso da câmera com o roboRIO, consulte a seção :ref: 'Vision Processing <docs/software/vision-processing/index:Vision Processing>' deste documento.

1.3.9 OpenMesh OM5P-AN or OM5P-AC Radio



Ambos os OpenMesh OM5P-AN e OpenMesh OM5P-AC wireless radio são utilizados como o

rádio do robô para fornecer funcionalidade de comunicação wireless ao robô. O dispositivo pode ser configurado como um ponto de acesso para conexão direta de um laptop para uso doméstico. Também pode ser configurado como Bridge para uso em campo. O rádio do robô deve ser alimentado por uma das saídas de 12V / 2A no VRM e conectado ao controlador roboRIO por Ethernet. Para mais informações, acesse [Programando seu Rádio](#).

O OM5P-AN [não está mais disponível para compra](#). O OM5P-AC é um pouco mais pesado, possui mais grades de resfriamento e possui uma textura superficial áspera em comparação com o OM5P-AN.

1.3.10 120A Circuit Breaker



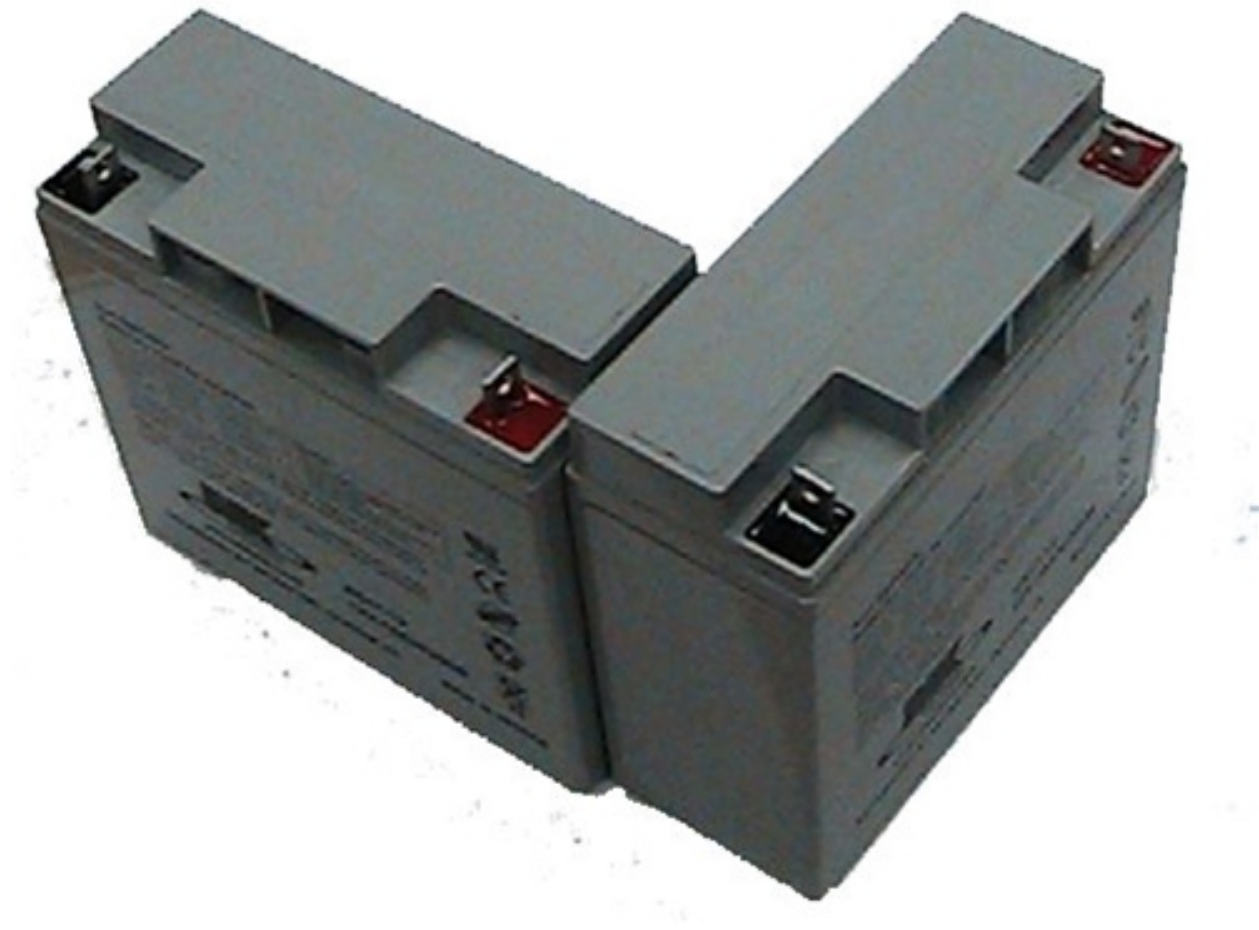
O 120A Main Circuit Breaker oferece duas funções no robô: a principal power switch do robô e um dispositivo de proteção para a fiação e os componentes do robô. O disjuntor 120A é conectado aos terminais positivos da bateria do robô e dos quadros de distribuição de energia. Para mais informações, acesse a [Ficha de dados Cooper Bussmann 18X Series \(PN: 185120F\)](#)

1.3.11 Snap Action Circuit Breakers



As snap Action circuit breakers, MX5-A40 and VB3 series, são usadas com a Power Distribution Panel para limitar a corrente aos circuitos de derivação. A MX5-A40 40A MAXI style circuit breaker é usada com os canais maiores na Power Distribution Panel para alimentar cargas que consomem corrente de até 40A. A VB3 series são usados com os canais menores no PDP para alimentar os circuitos com corrente de 30 A ou menos. Para mais informações, acesse as fichas de dados para a [MX5 series](#) e [VB3 Series](#).

1.3.12 Robot Battery



A fonte de alimentação de um robô de FRC é uma única bateria de 12V 18Ah. As baterias usadas para FRC são baterias de chumbo-ácido seladas, capazes de atender às altas demandas de corrente de um robô FRC. Para obter mais informações, consulte os dados para o [MK ES17-12](#) e [nersys NP18-12](#).

Note: Outros números de peça da bateria podem ser legais, consulte o [Manual FRC](#) para uma lista completa.

1.3.13 Crédito das Imagens

Imagem do roboRIO, cortesia da National Instruments. Imagens do Talon SRX, Victor 888, Victor SP, Victor SPX, e Spike H-Bridge Relay cortesia da VEX Robotics, Inc. Imagem do SPARK MAX cortesia da REV Robotics. Imagens da Lifecam, PDP, PCM, SPARK, e VRM cortesia da *FIRST*®. Todas as outras fotos são cortesia da AndyMark Inc.

1.4 Offline Installation Preparation

This article contains instructions/links to components you will want to gather if you need to do offline installation of the FRC® Control System software.

1.4.1 Documentation

This documentation can be downloaded for offline viewing. The link to download the PDF can be found [here](#).

1.4.2 Installers

All Teams

- [2020 FRC Game Tools](#) (Note: Click on link for “Individual Offline Installers”)
- [2020 FRC Radio Configuration Utility](#) or [2020 FRC Radio Configuration Utility Israel Version](#)
- (Optional - Veterans Only!) [Classmate/Acer PC Image](#)

LabVIEW Teams

- LabVIEW USB (from *FIRST*® Choice) or [Download](#) (Note: Click on link for “Individual Offline Installers”)

C++/Java Teams

- [C++/Java WPILib Installer](#)
- Visual Studio Code (if using Windows, run the installer and use it to download the appropriate VS Code file. If using macOS/Linux, download Visual Studio Code from [here](#))

1.4.3 3rd Party Libraries/Software

A number of software components were broken out of WPILib in 2017 and are now maintained by third parties. See [this blog](#) for more details.

A directory of available 3rd party software that plugs in to WPILib can be found on [docs/software/wpilib-overview/3rd-party-libraries:3rd Party Libraries](#).

1.5 Instalando LabVIEW para FRC (LabVIEW apenas)



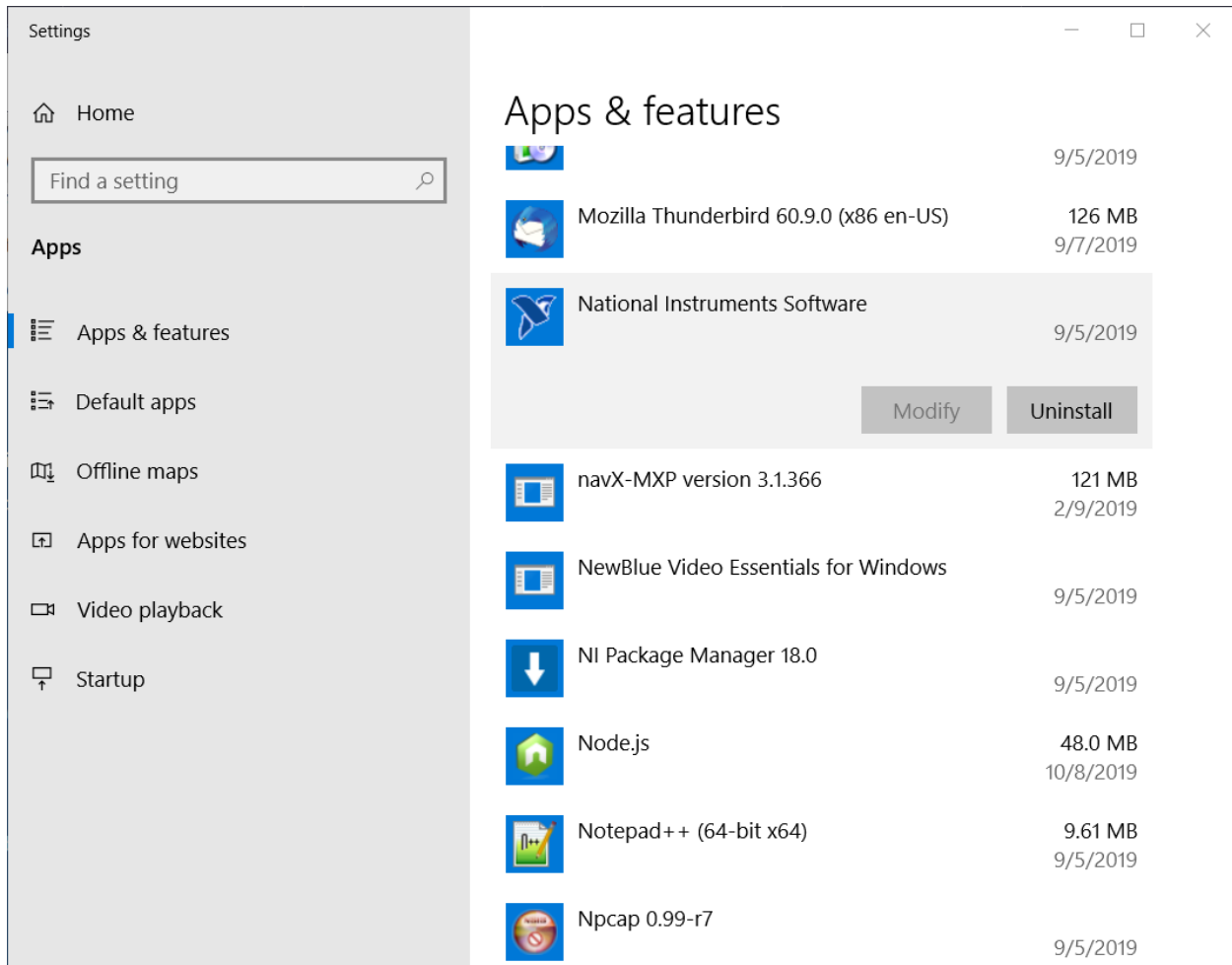
Note: Esta instalação é para times que programam em LabVIEW ou usam o NI Vision Assistant apenas. Times que utilizam C++ e Java que não estão utilizando estes recursos não precisam instalar a partir do DVD e devem prosseguir para [Instalando FRC Game Tools](#).

O tempo para download e instalação varia muito de acordo com as especificações de conexão do computador, no entanto, note que esse processo envolve o download e a instalação de um arquivo grande e provavelmente levará pelo menos uma hora para ser concluído.

1.5.1 Desinstale as Versões Antigas (Recomendado)

Note: Se você deseja continuar utilizando cRIOs você precisará manter uma instalação do LabVIEW para FRC 2014. A licença para o LabVIEW para FRC 2014 foi estendida. Embora essas versões possam coexistir em um único computador, essa não é uma configuração extensivamente testada.

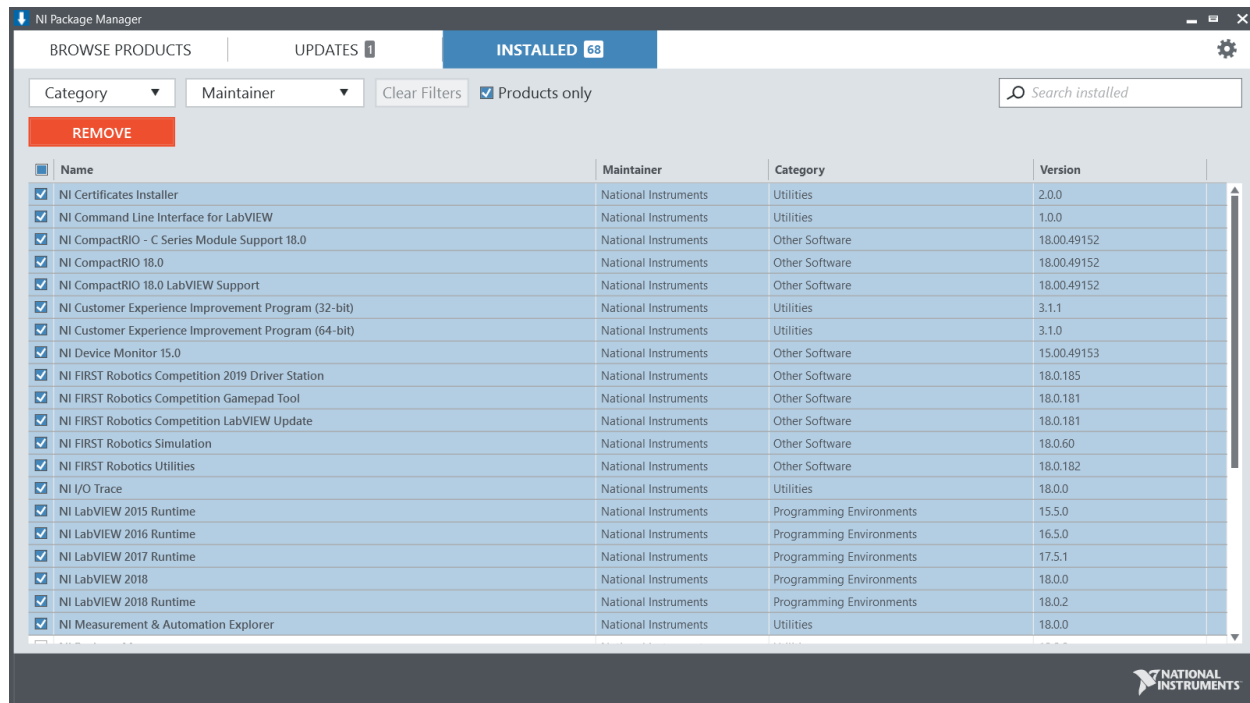
Antes de instalar a nova versão do LabVIEW, é recomendado remover as versões antigas. A nova versão provavelmente coexistirá com a versão antiga, porém todos os testes foram feitos apenas com a FRC 2020. Certifique-se de fazer um backup de qualquer código do time localizado no diretório “User\LabVIEW Data” antes de desinstalar. Em seguida clique em Iniciar >> Adicionar ou Remover Programas. Localize “National Instruments Software” e selecione “Uninstall”.



Selecione Componentes para Desinstalar

Na caixa de diálogo exibida, selecione todas as entradas. A maneira mais fácil de fazer isso é desmarcar a caixa de seleção “Products Only” e marcar a caixa de seleção à esquerda de “Name”. Clique em “Remove”. Aguarde o desinstalador concluir e, se solicitado, reinicie o computador.

Warning: Essas instruções assumem que nenhum outro software da National Instruments esteja instalado. Se você tiver outro software da National Instruments instalado, é necessário desmarcar o software que não deve ser desinstalado.



1.5.2 Getting LabVIEW installer

Either locate and insert the LabVIEW USB Drive or download the LabVIEW 2020 installer from https://www.ni.com/en-us/support/downloads/drivers/download_labview-software-for-frc.html

DOWNLOAD

Online installer

File Size

5.37 MB

Note:

If you need to download individual versions or patches, you can select from Individual Offline Installers

Offline Installer

If you wish to install on other machines offline, do not click the Download button, click **Individual Offline Installers** and then click Download, to download the full installer.

Note: This is a large download (~8GB). It is recommended to use a fast internet connection and to use the NI Downloader to allow the download to resume if interrupted.

1.5.3 Installing LabVIEW

National Instruments LabVIEW requires a license. Each season's license is active until January 31st of the following year (e.g. the license for the 2020 season expires on January 31, 2021)

Teams are permitted to install the software on as many team computers as needed, subject to the restrictions and license terms that accompany the applicable software, and provided that only team members or mentors use the software, and solely for FRC. Rights to use LabVIEW are governed solely by the terms of the license agreements that are shown during the installation of the applicable software.

Welcome

Starting Install

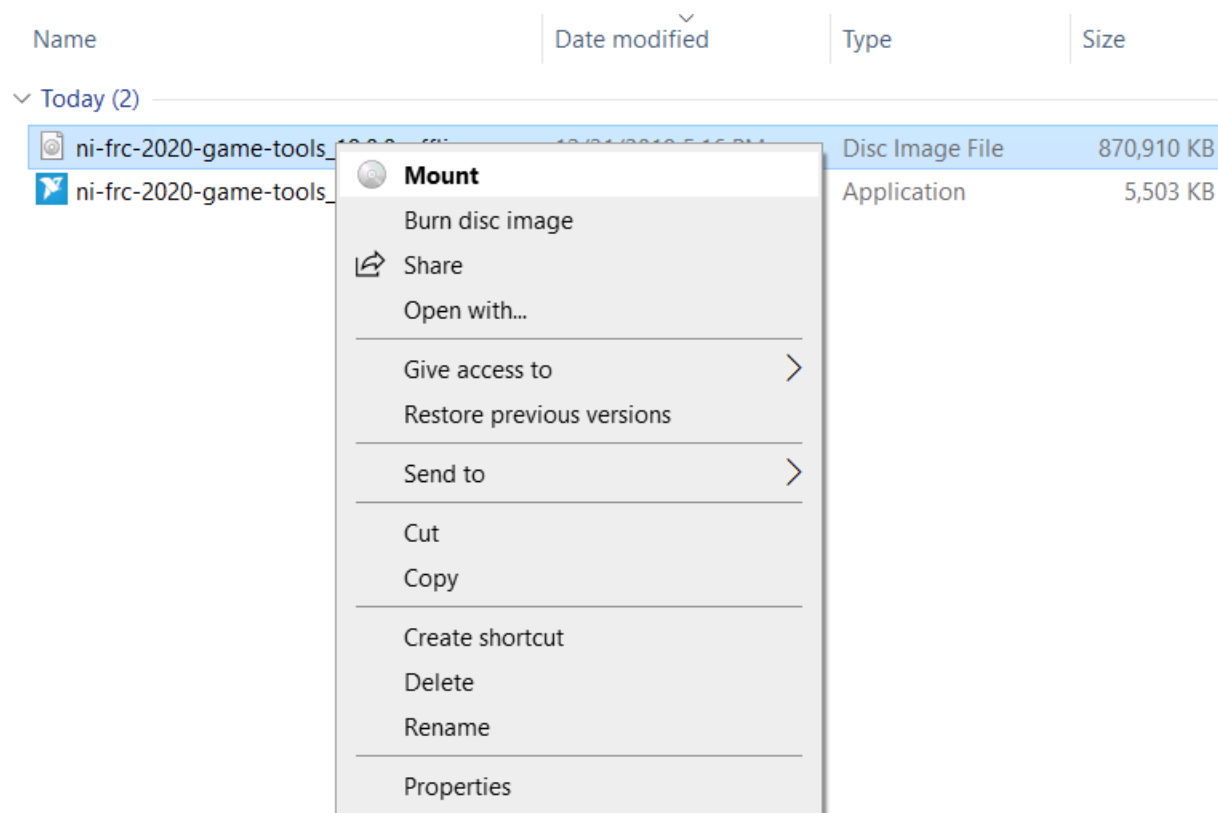
Online Installer

Run the downloaded exe file to start the install process. Click "Yes" if a Windows Security prompt

Offline Installer (Windows 10)

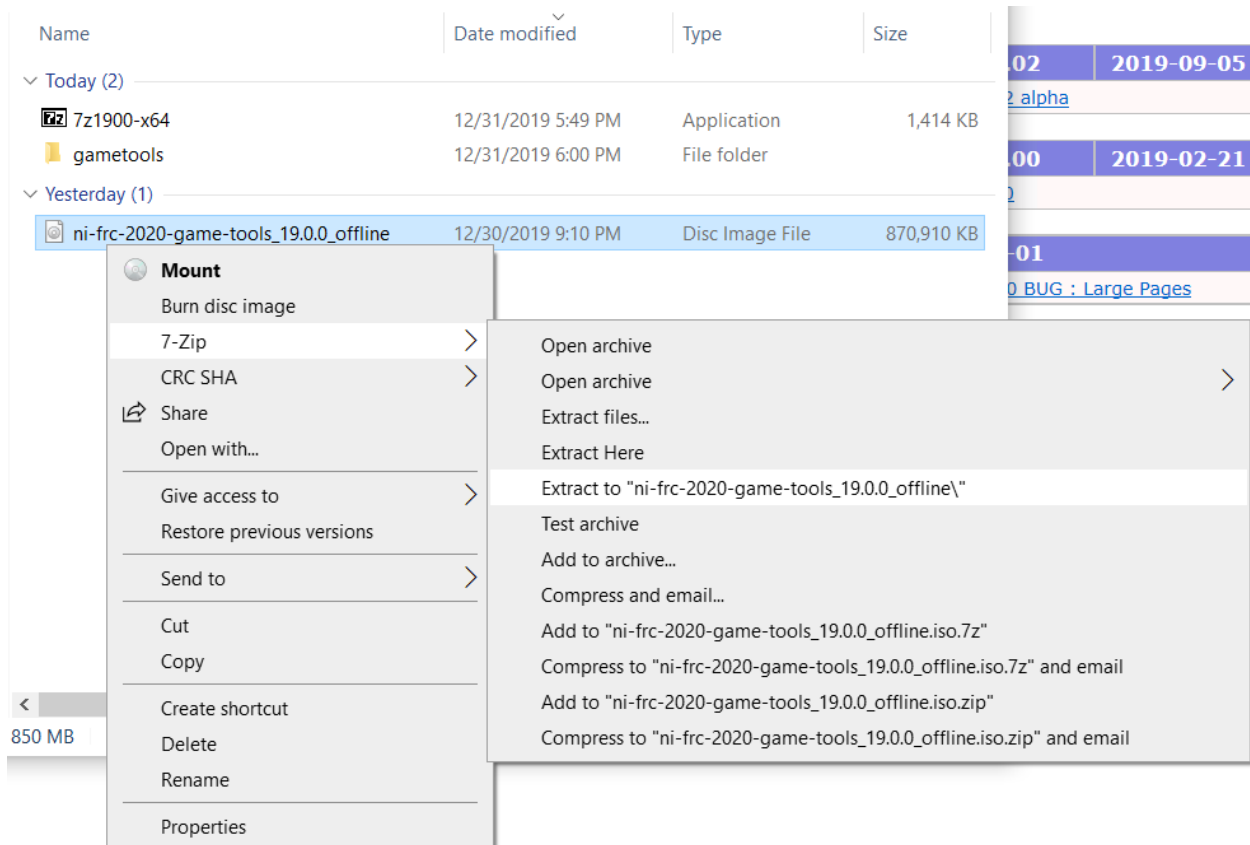
Right click on the downloaded iso file and select mount. Run install.exe from the mounted iso. Click "Yes" if a Windows Security prompt

Note: other installed programs may associate with iso files and the mount option may not appear. If that software does not give the option to mount or extract the iso file, then follow the directions in the "Offline Installer (Windows 7, 8, & 8.1)" tab.

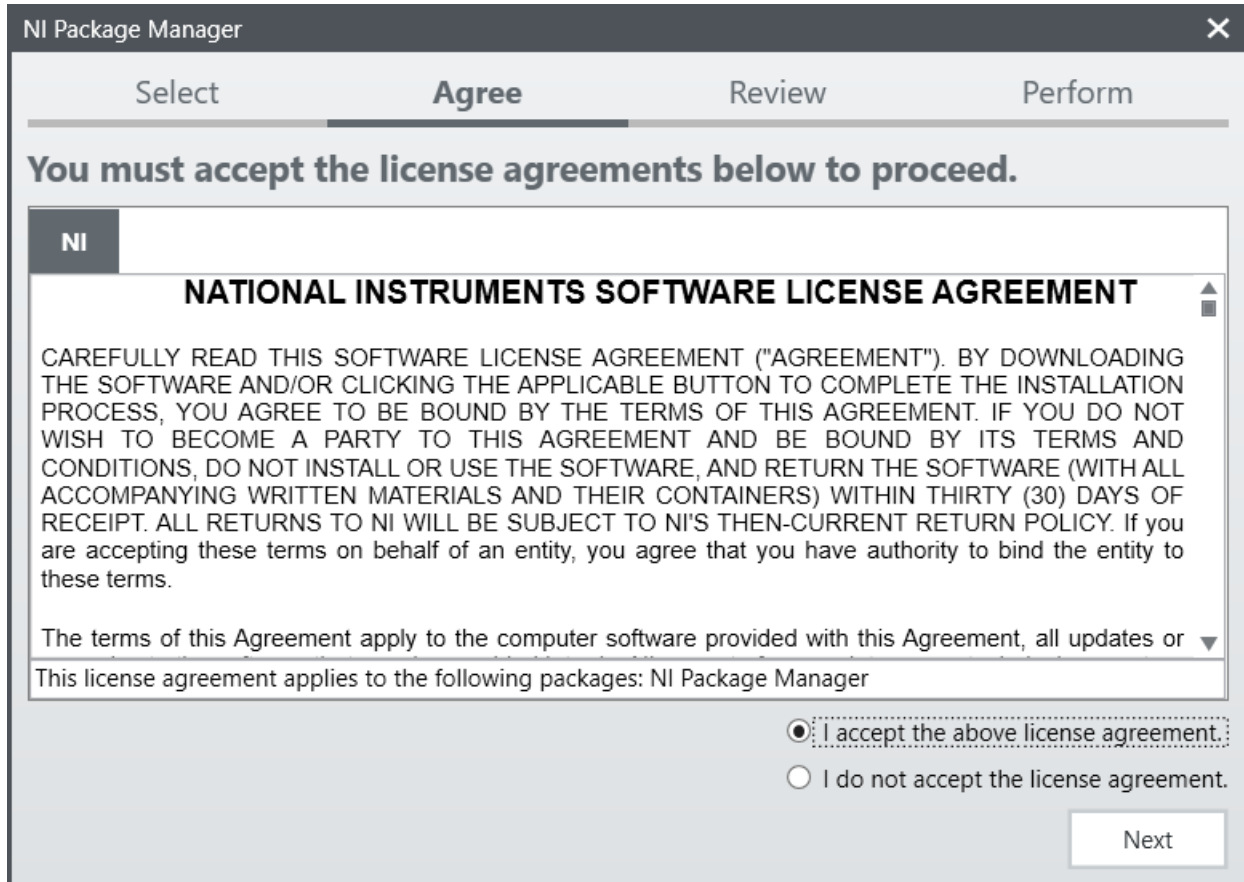


Offline Installer (Windows 7, 8, & 8.1)

Install 7-Zip (download [here](#)). As of the writing of this document, the current released version is 19.00 (2019-02-21). Right click on the downloaded iso file and select Extract to.



Run install.exe from the extracted folder. Click “Yes” if a Windows Security prompt appears. Click “Yes” if a Windows Security prompt appears.

NI Package Manager License

The image shows a screenshot of the 'NI Package Manager' window. At the top, there is a title bar with the text 'NI Package Manager' and a close button. Below the title bar is a navigation bar with four tabs: 'Select', 'Agree', 'Review', and 'Perform'. The 'Agree' tab is currently selected. The main content area has a header that says 'You must accept the license agreements below to proceed.' Below this header is a section titled 'NI' with a sub-header 'NATIONAL INSTRUMENTS SOFTWARE LICENSE AGREEMENT'. The text of the agreement is displayed in a scrollable area. At the bottom of the window, there are two radio buttons: 'I accept the above license agreement.' (which is selected) and 'I do not accept the license agreement.' To the right of these radio buttons is a 'Next' button.

NI Package Manager

Select Agree Review Perform

You must accept the license agreements below to proceed.

NI

NATIONAL INSTRUMENTS SOFTWARE LICENSE AGREEMENT

CAREFULLY READ THIS SOFTWARE LICENSE AGREEMENT ("AGREEMENT"). BY DOWNLOADING THE SOFTWARE AND/OR CLICKING THE APPLICABLE BUTTON TO COMPLETE THE INSTALLATION PROCESS, YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT WISH TO BECOME A PARTY TO THIS AGREEMENT AND BE BOUND BY ITS TERMS AND CONDITIONS, DO NOT INSTALL OR USE THE SOFTWARE, AND RETURN THE SOFTWARE (WITH ALL ACCOMPANYING WRITTEN MATERIALS AND THEIR CONTAINERS) WITHIN THIRTY (30) DAYS OF RECEIPT. ALL RETURNS TO NI WILL BE SUBJECT TO NI'S THEN-CURRENT RETURN POLICY. If you are accepting these terms on behalf of an entity, you agree that you have authority to bind the entity to these terms.

The terms of this Agreement apply to the computer software provided with this Agreement, all updates or

This license agreement applies to the following packages: NI Package Manager

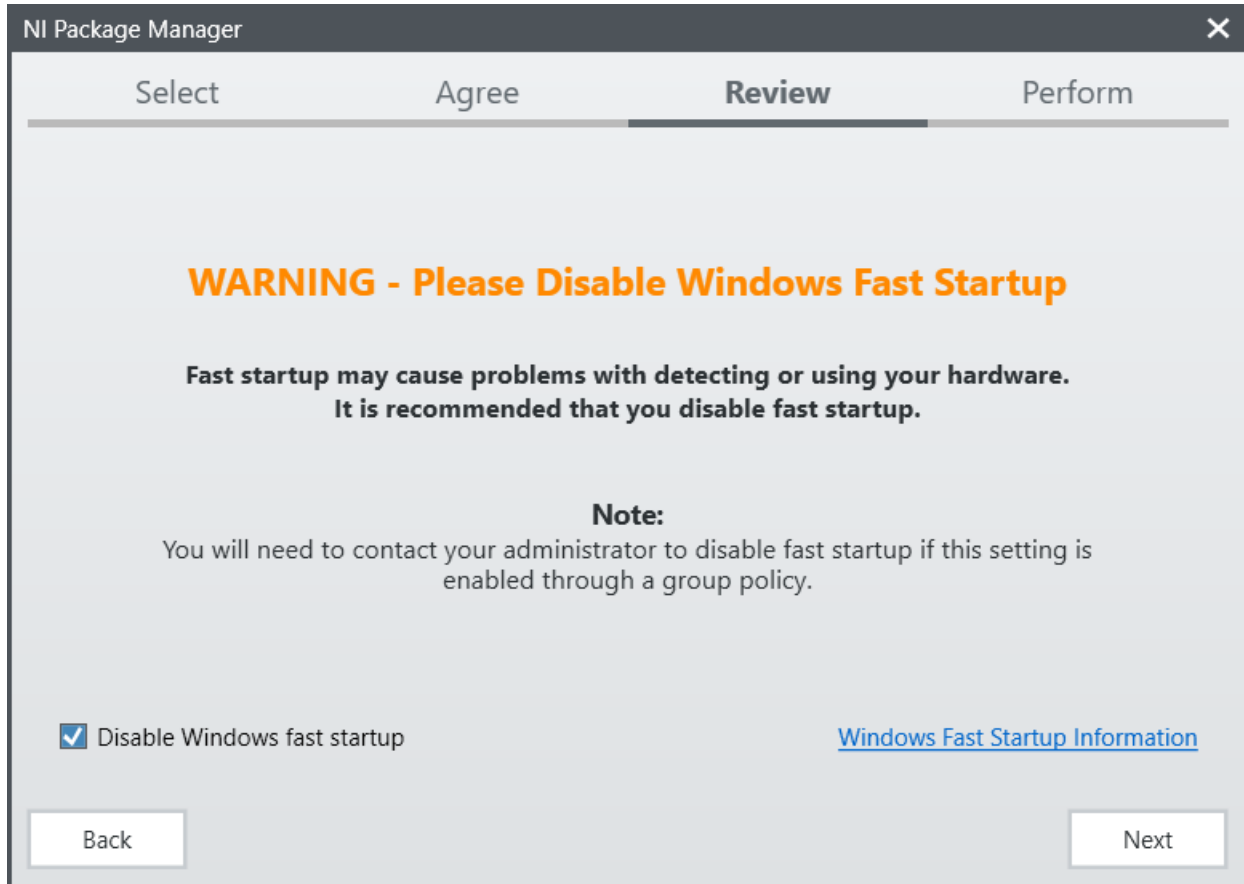
☒ I accept the above license agreement.

☐ I do not accept the license agreement.

Next

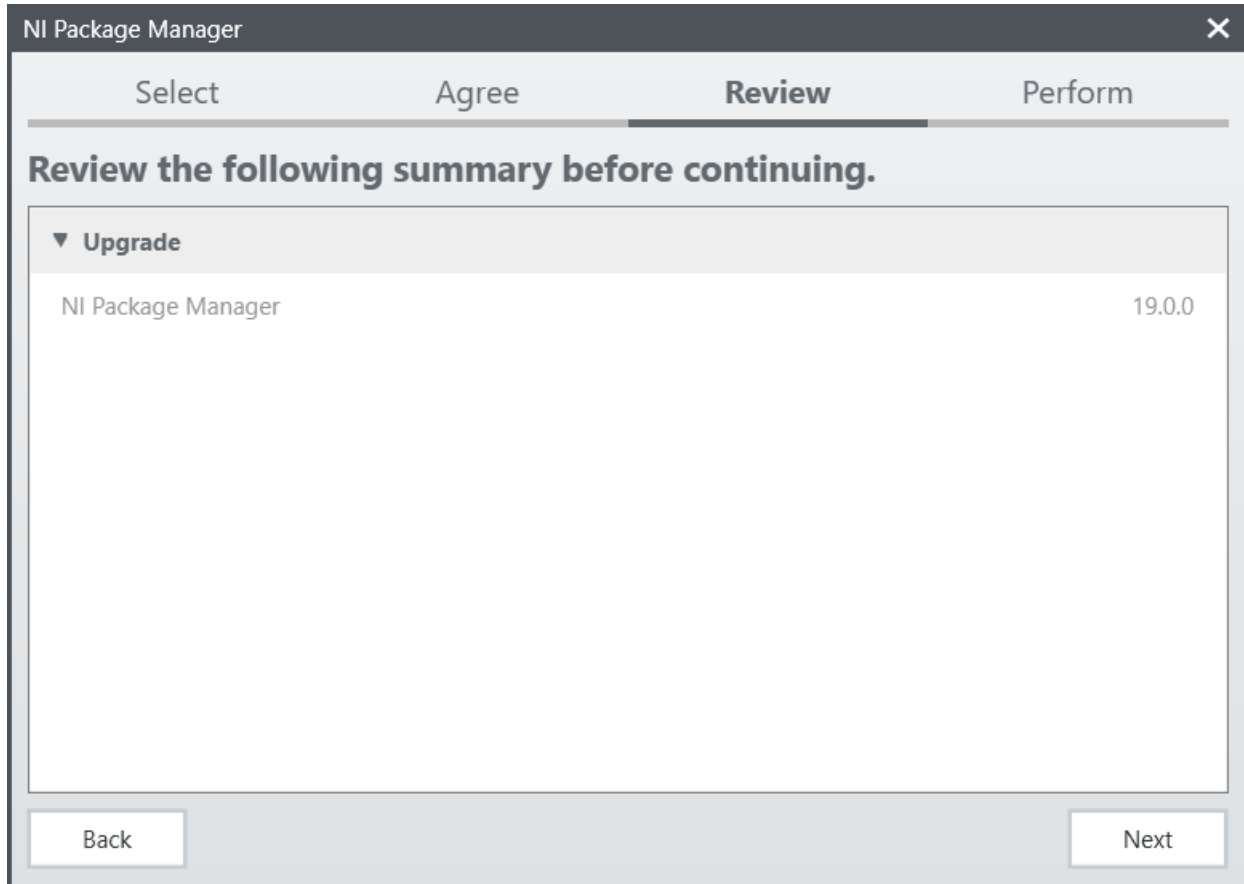
If you see this screen, click "Next"

Disable Windows Fast Startup



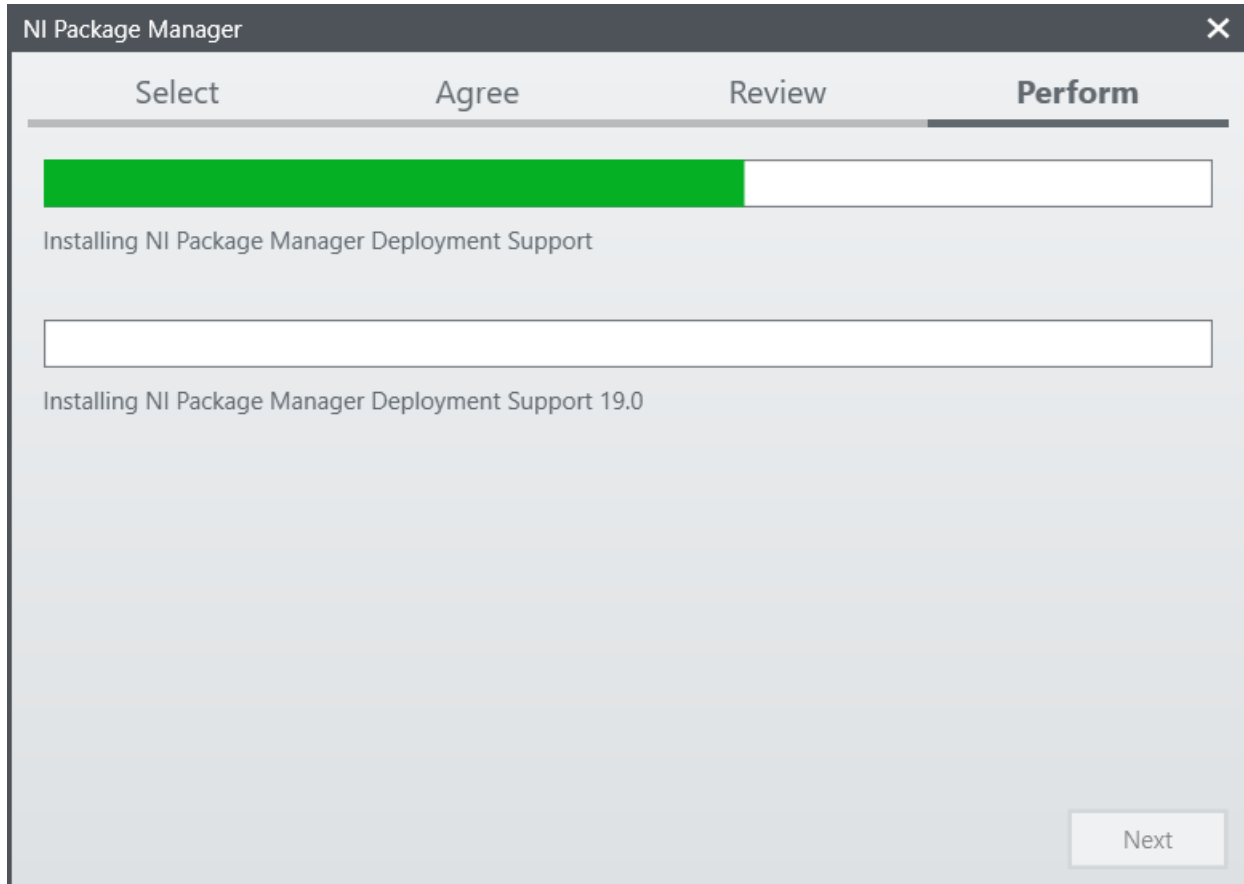
If you see this screen, click “Next”

NI Package Manager Review



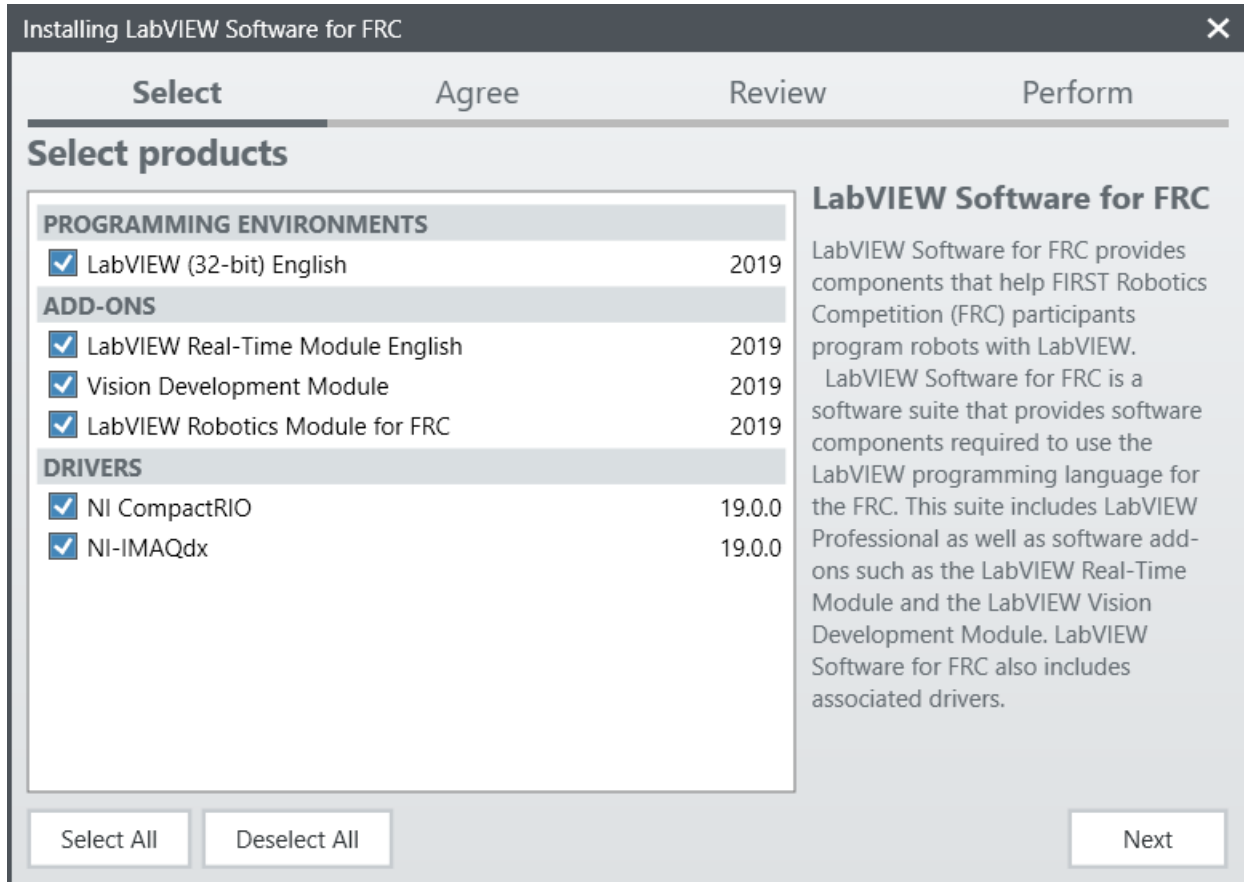
If you see this screen, click “Next”

NI Package Manager Installation



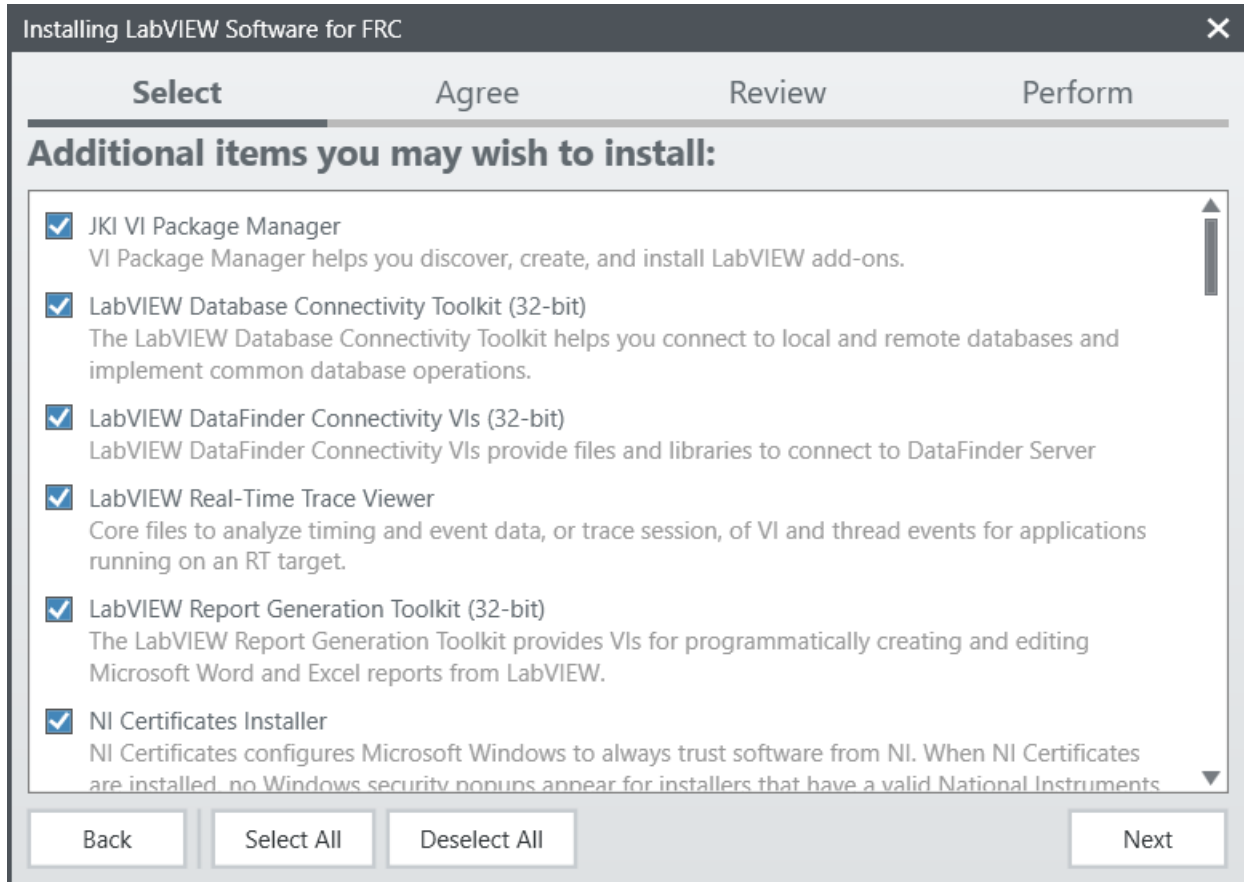
Installation progress of the NI Package Manager will be tracked in this window

Product List



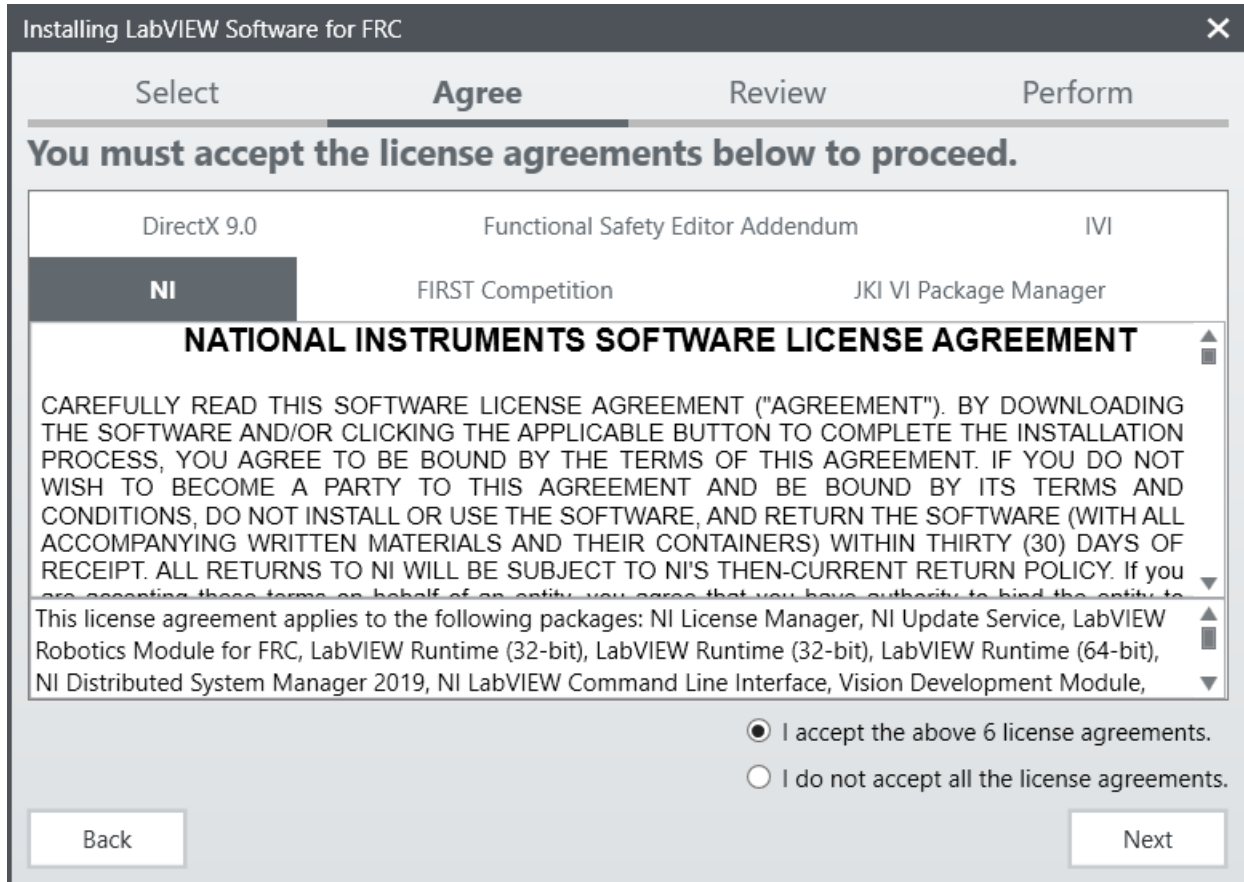
Click "Next"

Additional Packages

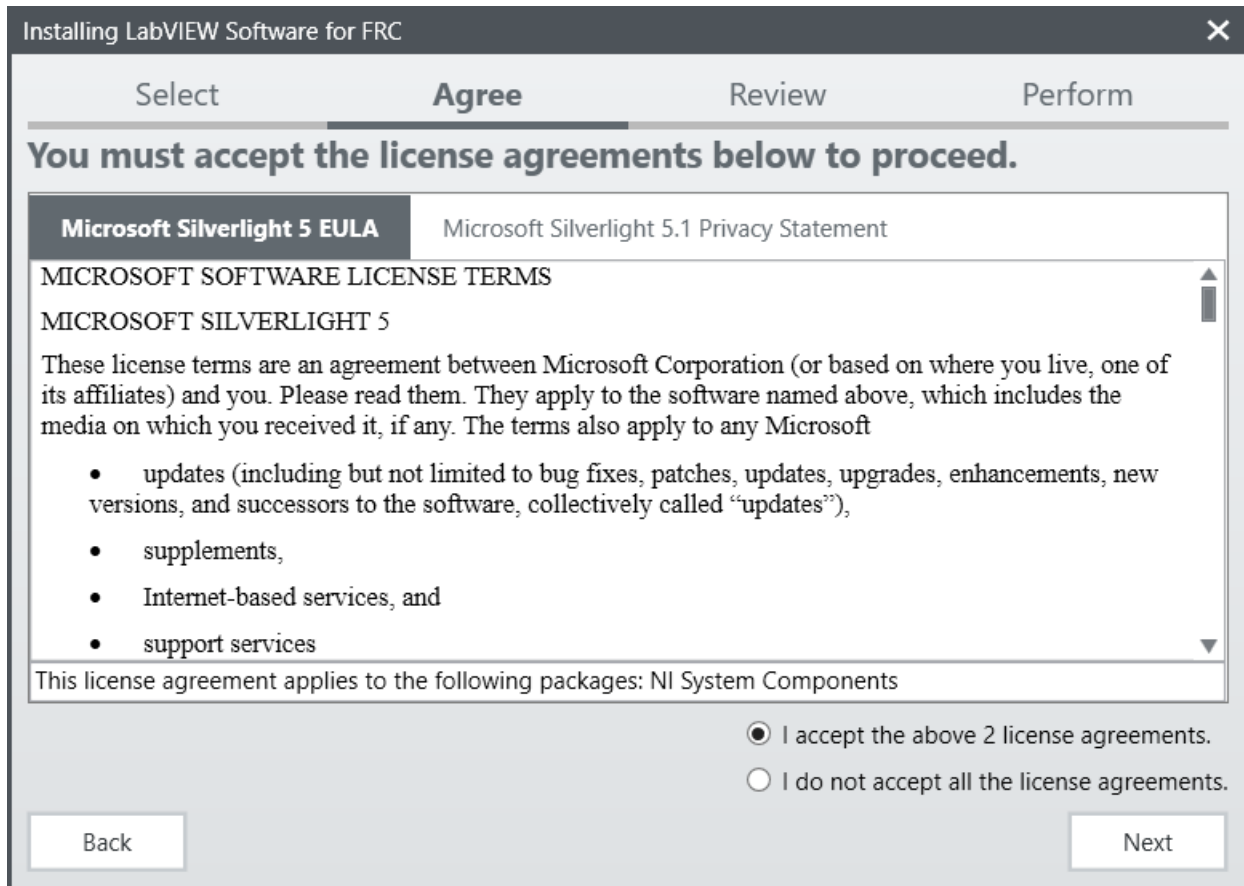


Click "Next"

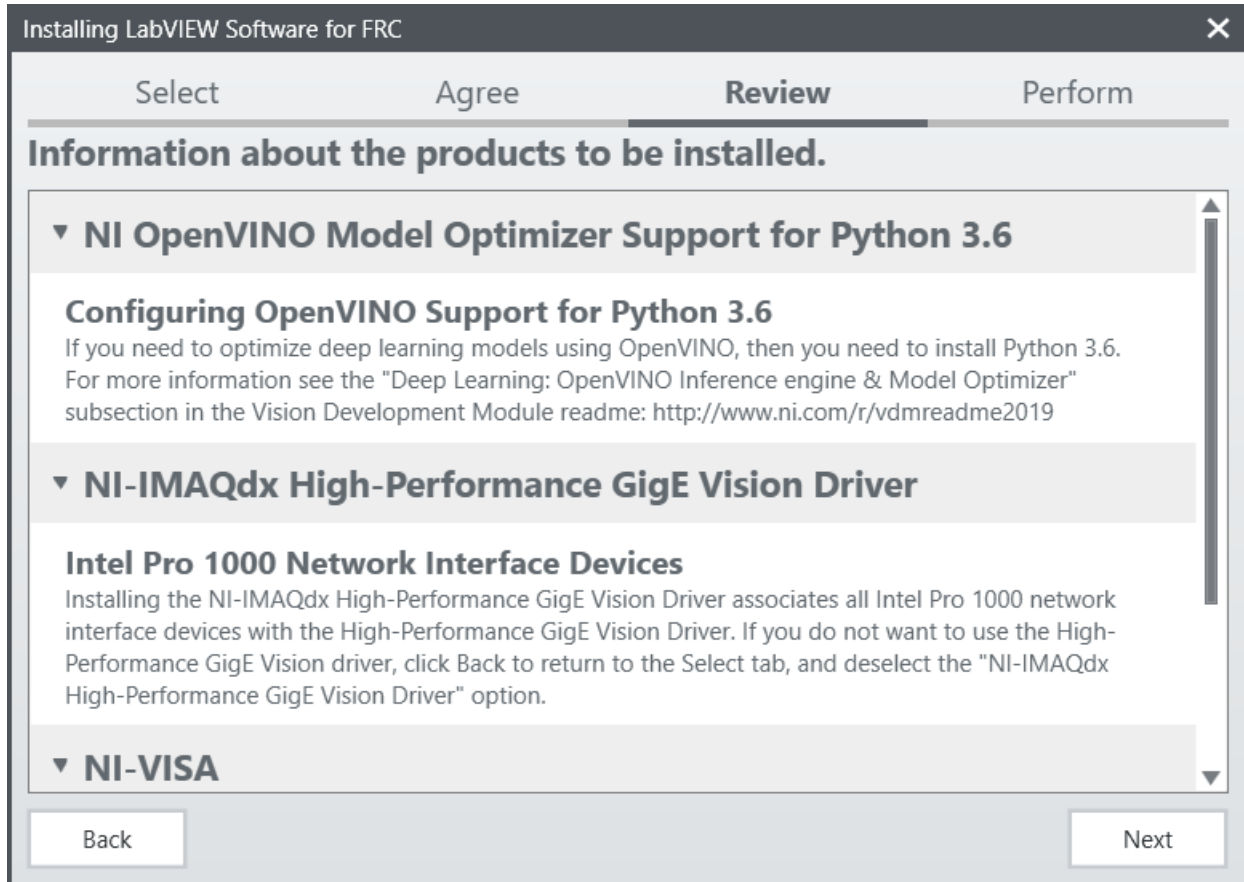
License agreements



Check "I accept..." then Click "Next"

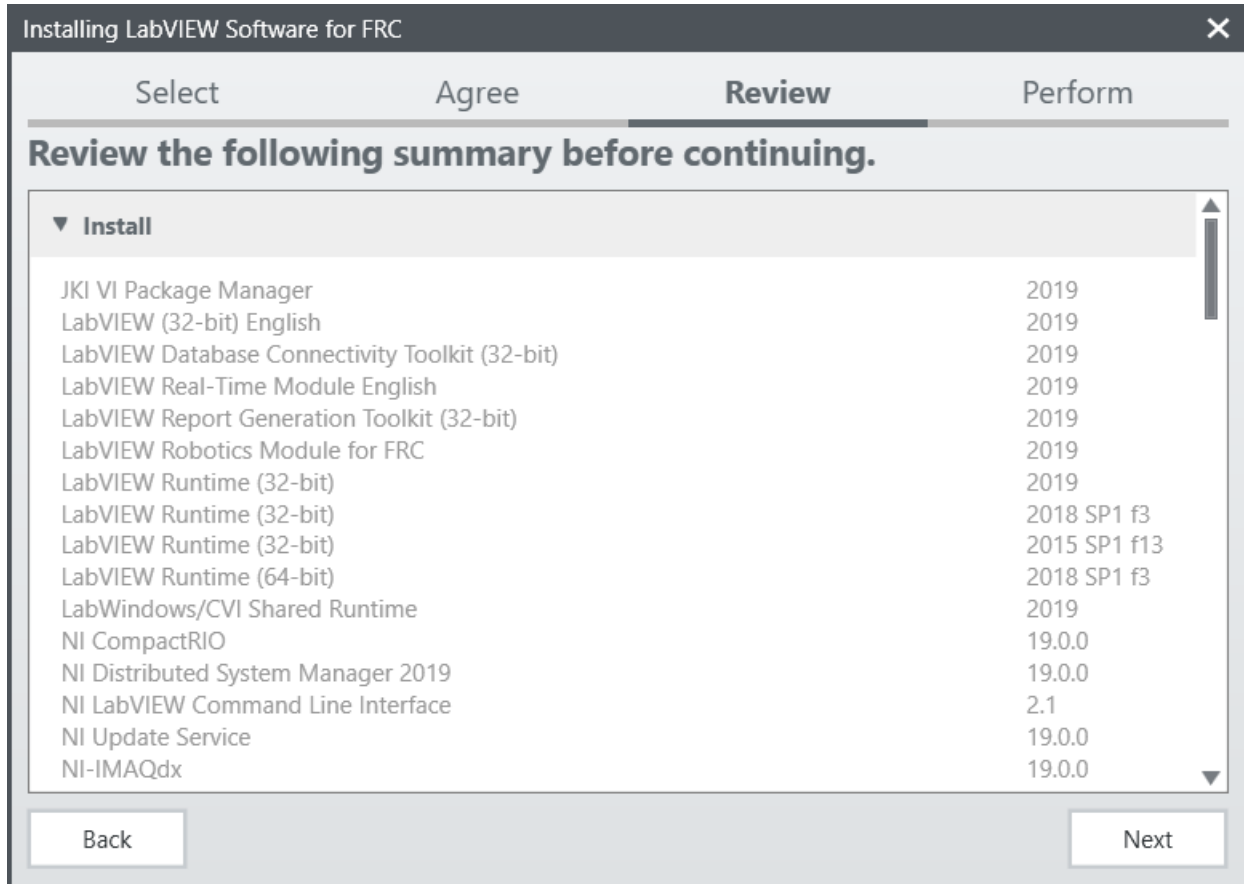


Check “I accept...” then Click “Next”

Product Information

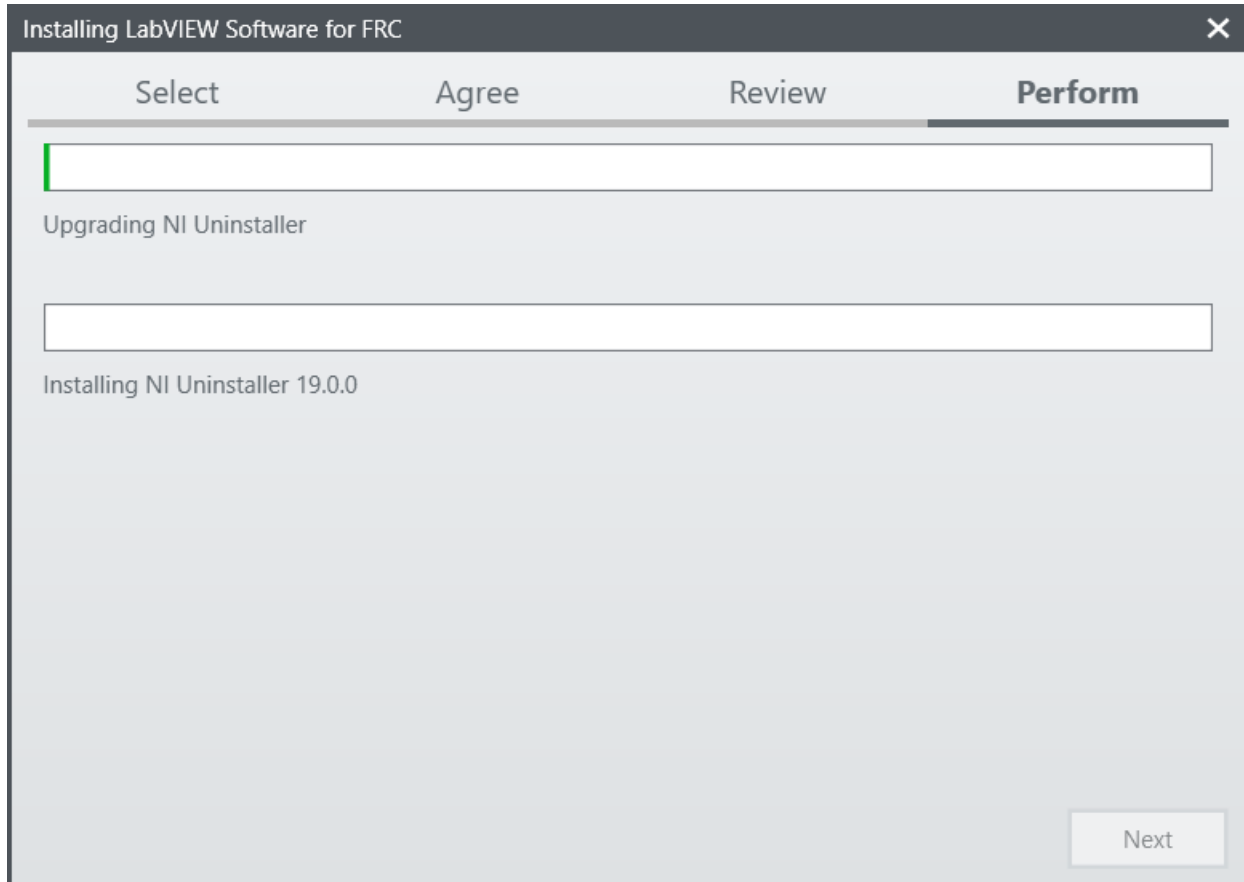
Click "Next"

Start Installation



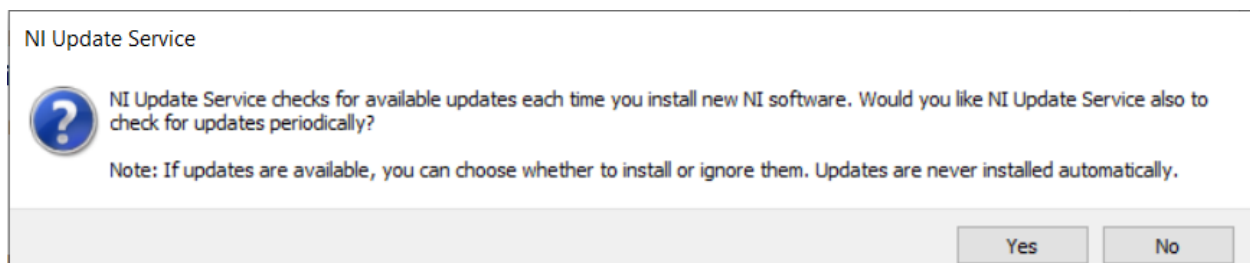
Click "Next"

Overall Progress



Overall installation progress will be tracked in this window

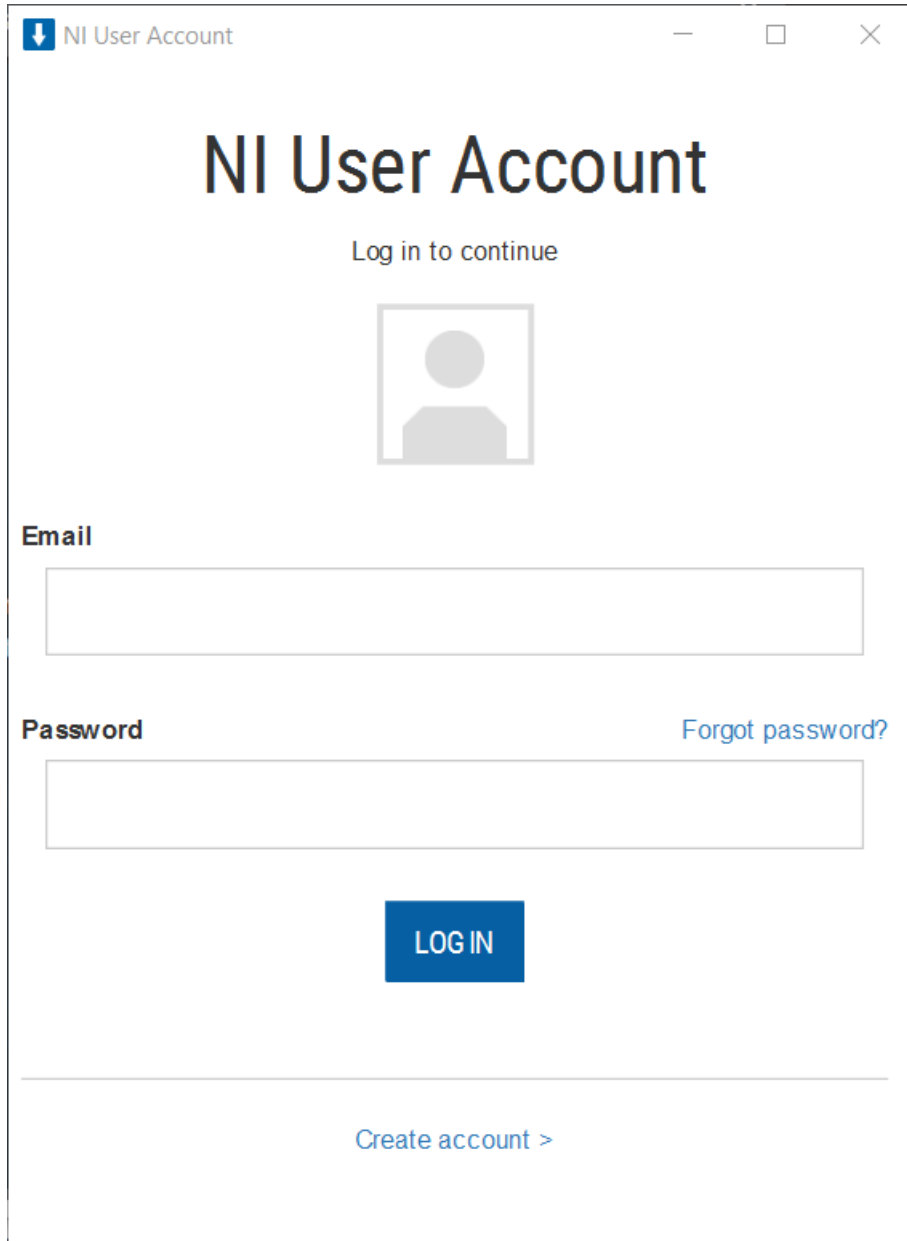
1.5.4 NI Update Service



You will be prompted whether to enable the NI update service. You can choose to not enable the update service.

Warning: It is not recommended to install these updates unless directed by FRC through our usual communication channels (FRC Blog, Team Updates or E-mail Blasts).

NI Activation Wizard



The screenshot shows a window titled "NI User Account" with standard Windows window controls (minimize, maximize, close). The main heading is "NI User Account" in a large, bold font. Below it is the text "Log in to continue". There is a placeholder for a user profile picture. Below the picture are two input fields: "Email" and "Password". To the right of the "Password" field is a link that says "Forgot password?". Below the input fields is a blue "LOGIN" button. At the bottom of the window, there is a link that says "Create account >".

Log into your ni.com account. If you don't have an account, select 'Create account' to create a free account.

Serial Number	
XXXXXXXXXX	LabVIEW 2019 Application Builder
XXXXXXXXXX	LabVIEW 2019 Base Development System
XXXXXXXXXX	LabVIEW 2019 Continuous Integration System
XXXXXXXXXX	LabVIEW 2019 Database Connectivity Toolkit
XXXXXXXXXX	LabVIEW 2019 Debug Deployment System
XXXXXXXXXX	LabVIEW 2019 Full Development System
XXXXXXXXXX	LabVIEW 2019 LabVIEW Robotics FRC Simulation
XXXXXXXXXX	LabVIEW 2019 Professional Development System
XXXXXXXXXX	LabVIEW 2019 Real-Time Debug Deployment

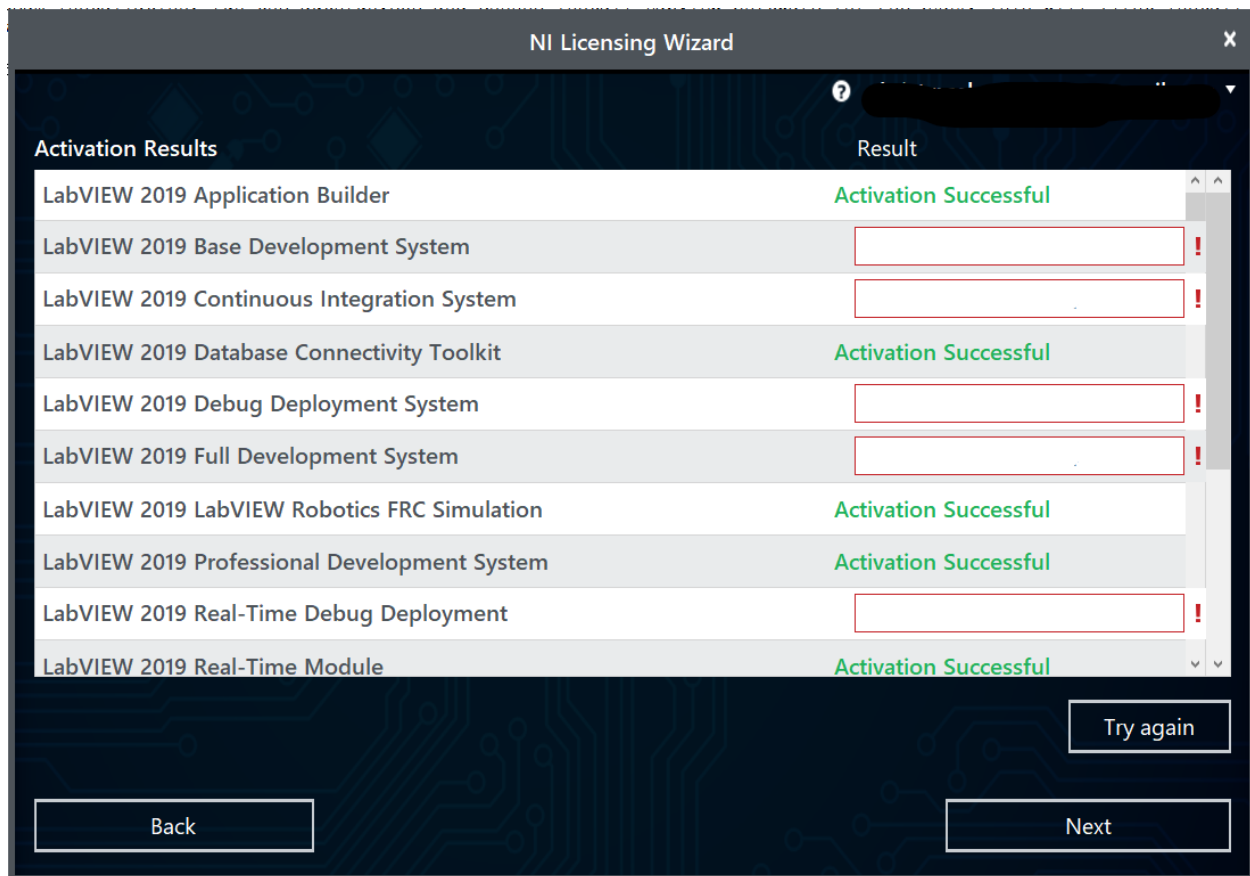
Enter Activation Codes

Refresh

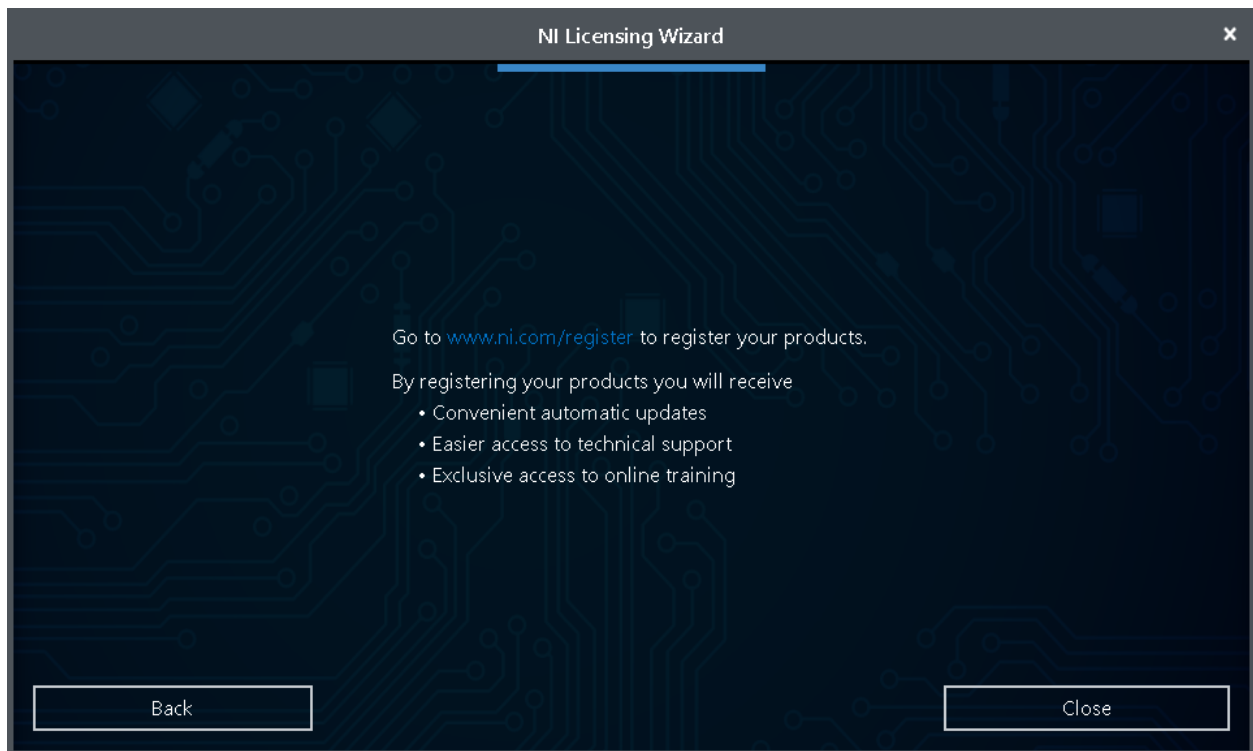
Activate

The serial number you entered at the “User Information” screen should appear in all of the text boxes, if it doesn’t, enter it now. Click “Activate”.

Note: If this is the first time activating the 2020 software on this account, you will see the message shown above about a valid license not being found. You can ignore this.

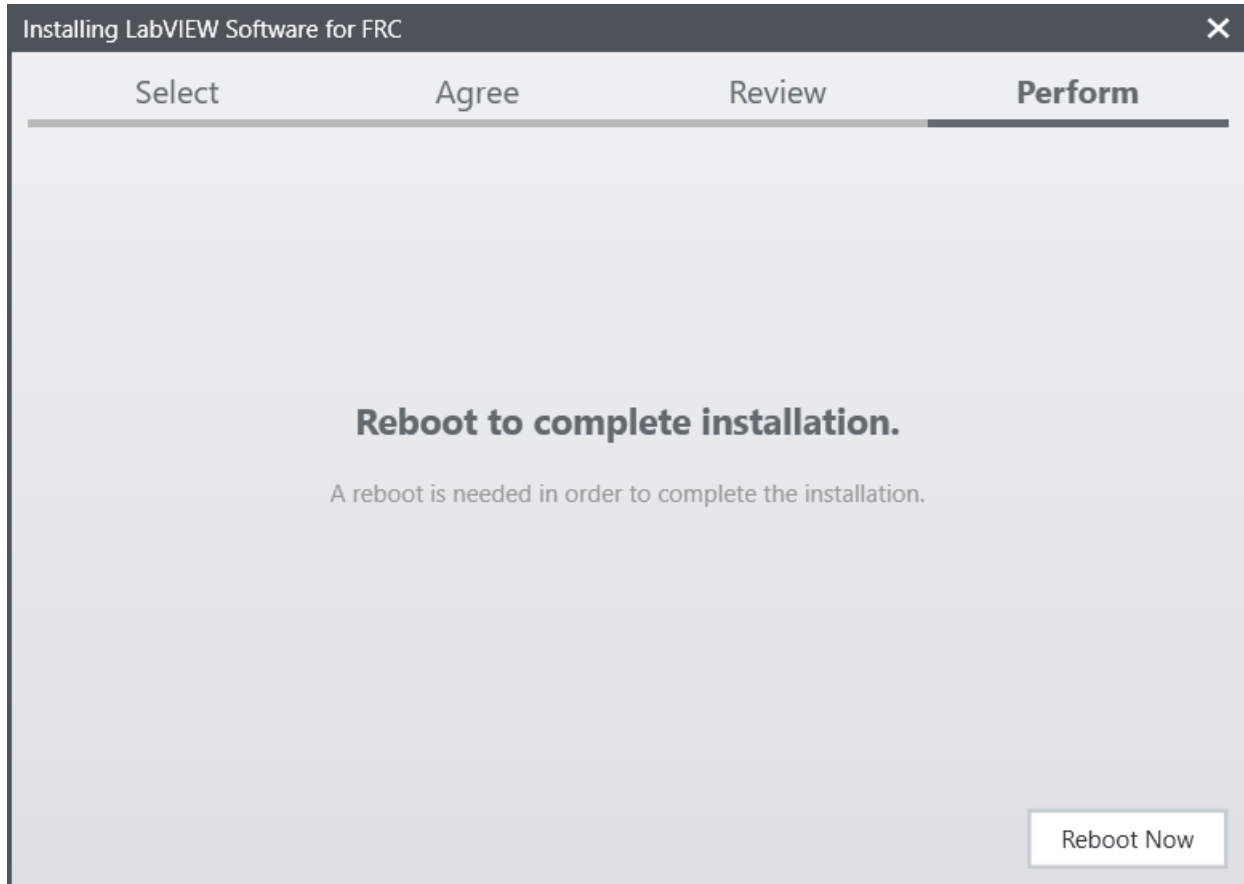


If your products activate successfully, an “Activation Successful” message will appear. If the serial number was incorrect, it will give you a text box and you can re-enter the number and select “Try Again”. The items shown above are not expected to activate. If everything activated successfully, click “Next”.



Click "Close".

Restart



Select "Reboot Now" after closing any open programs.

1.6 Installing the FRC Game Tools

The FRC Game Tools contains the following software components: LabVIEW Update, FRC Driver Station, and FRC Utilities. The LabVIEW runtime components required for the Driver Station and Utilities are included in this package. **No components from the LabVIEW Software for FRC package are required for running either the Driver Station or Utilities.**

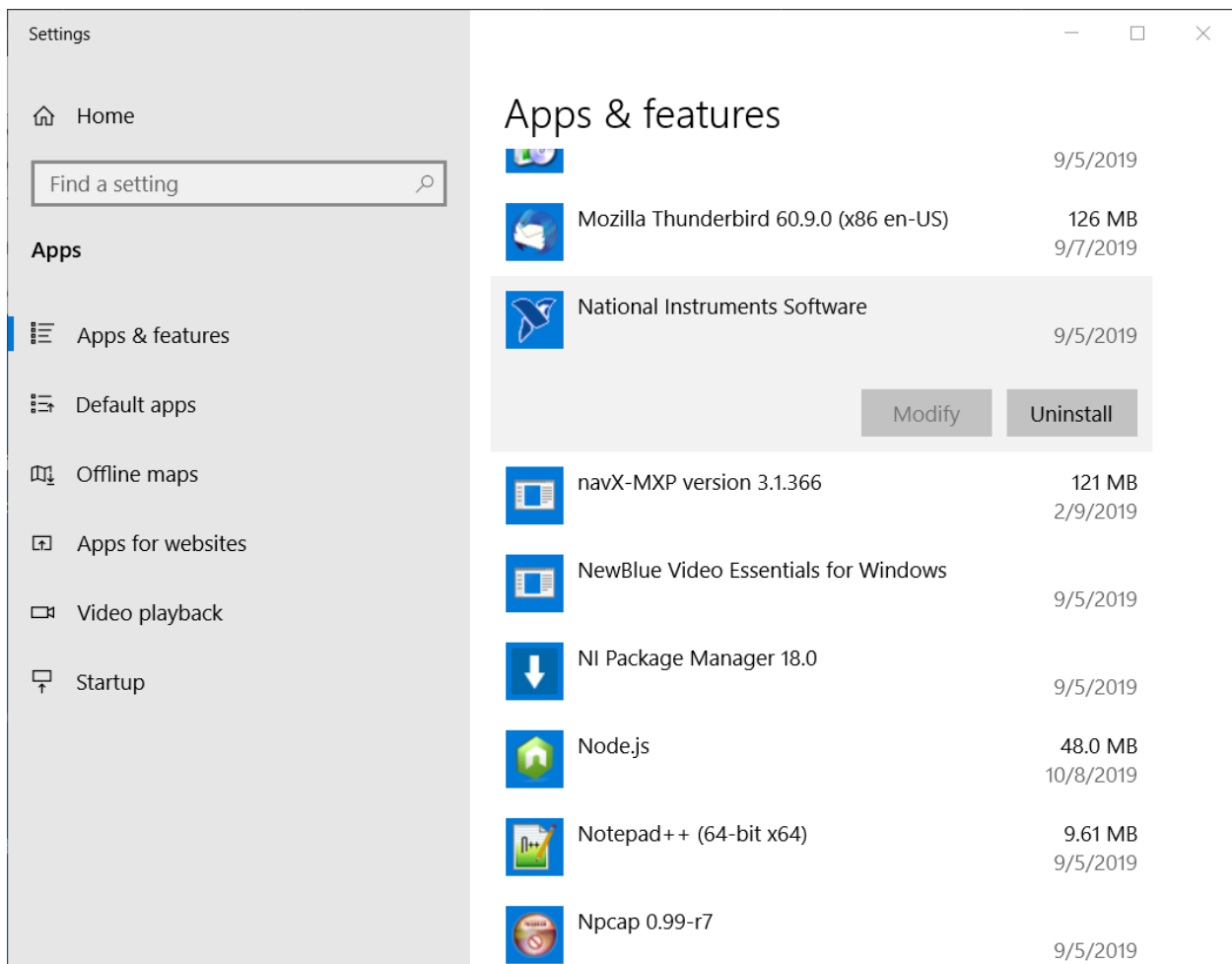
Note: The Driver Station will only work on Windows 7, Windows 8, Windows 8.1, and Windows 10. It will not work on Windows XP.

1.6.1 Uninstall Old Versions (Recommended)

Warning: LabVIEW teams have already completed this step, do not repeat it.

Note: It is only necessary to uninstall previous versions when installing a new year's tools. For example, uninstall the 2019 tools before installing the 2020 tools. It is not necessary to uninstall before upgrading to a new update of the 2020 game tools.

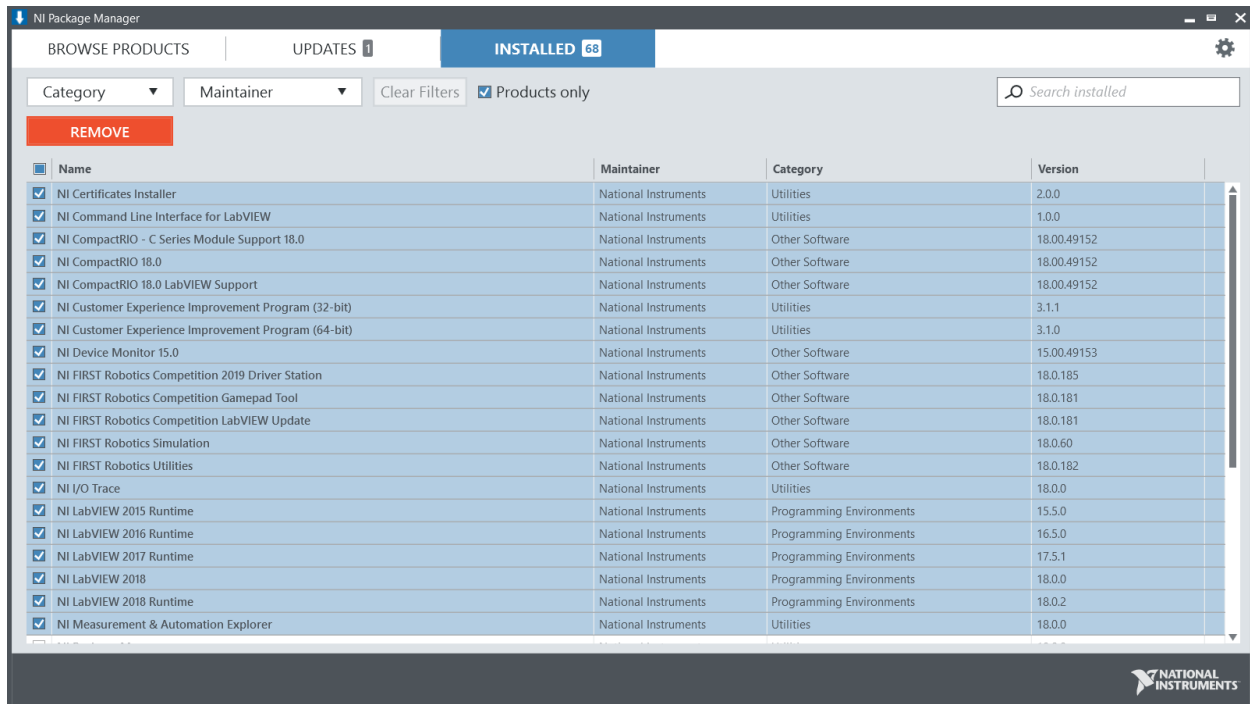
Before installing the new version of the FRC Game Tools it is recommended to remove any old versions. The new version will likely co-exist with the old version (note that the DS will overwrite old versions), but all testing has been done with FRC 2020 only. Then click Start >> Add or Remove Programs. Locate the entry labeled "National Instruments Software", and select Uninstall.



Select Components to Uninstall

In the dialog box that appears, select all entries. The easiest way to do this is to de-select the "Products Only" check-box and select the check-box to the left of "Name". Click Remove.

Wait for the uninstaller to complete and reboot if prompted.



1.6.2 Downloading the Update

Download the update from <https://www.ni.com/en-us/support/downloads/drivers/download.frc-game-tools.html>

DOWNLOAD

Online installer

File Size

5.37 MB

i **Note:** If you need to download individual versions or patches, you can select from Individual Offline Installers

Offline Installer

If you wish to install on other machines offline, do not click the Download button, click **Individual Offline Installers** and then click Download, to download the full installer.

1.6.3 .NET Framework 4.6.2

The Game Tools installer may prompt that .NET Framework 4.6.2 needs to be updated or installed. Follow prompts on-screen to complete the installation, including rebooting if re-

quested. Then resume the installation of the FRC Game Tools, restarting the installer if necessary.

1.6.4 Welcome

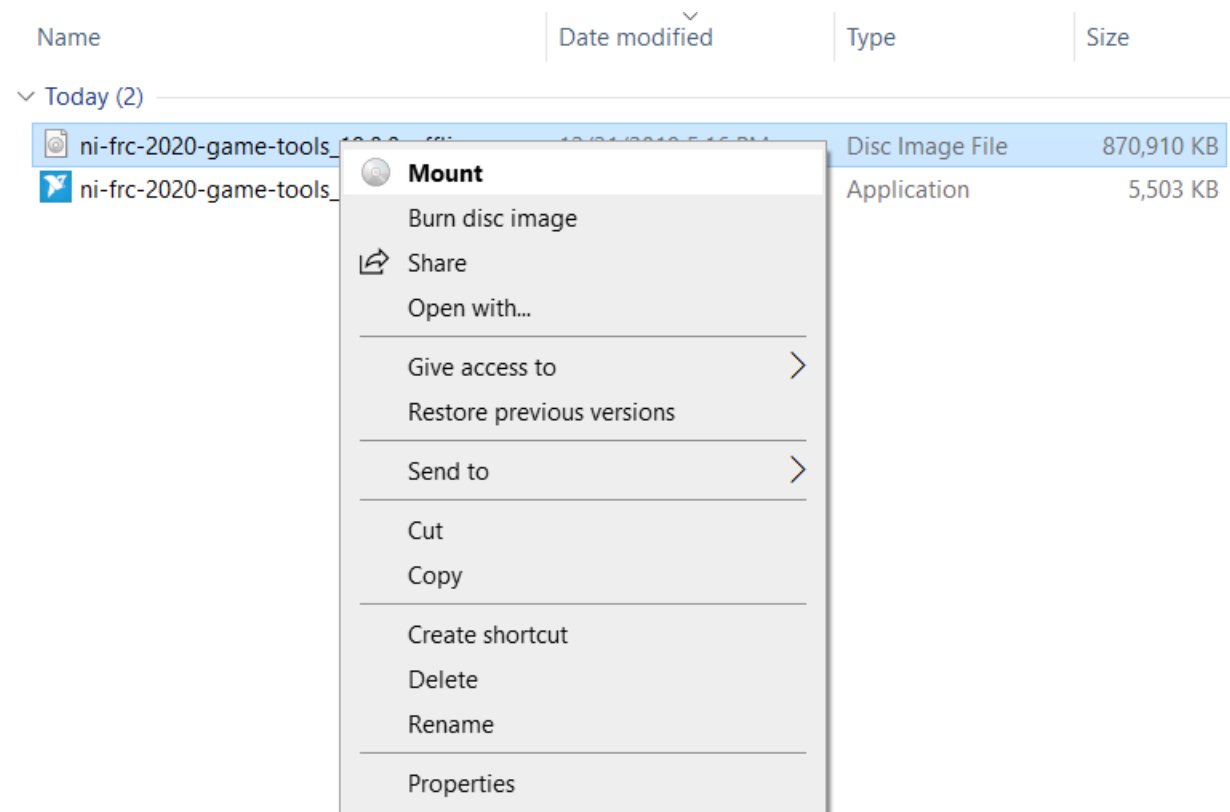
Starting Install

Online Installer

Run the downloaded exe file to start the install process. Click “Yes” if a Windows Security prompt

Offline Installer (Windows 10)

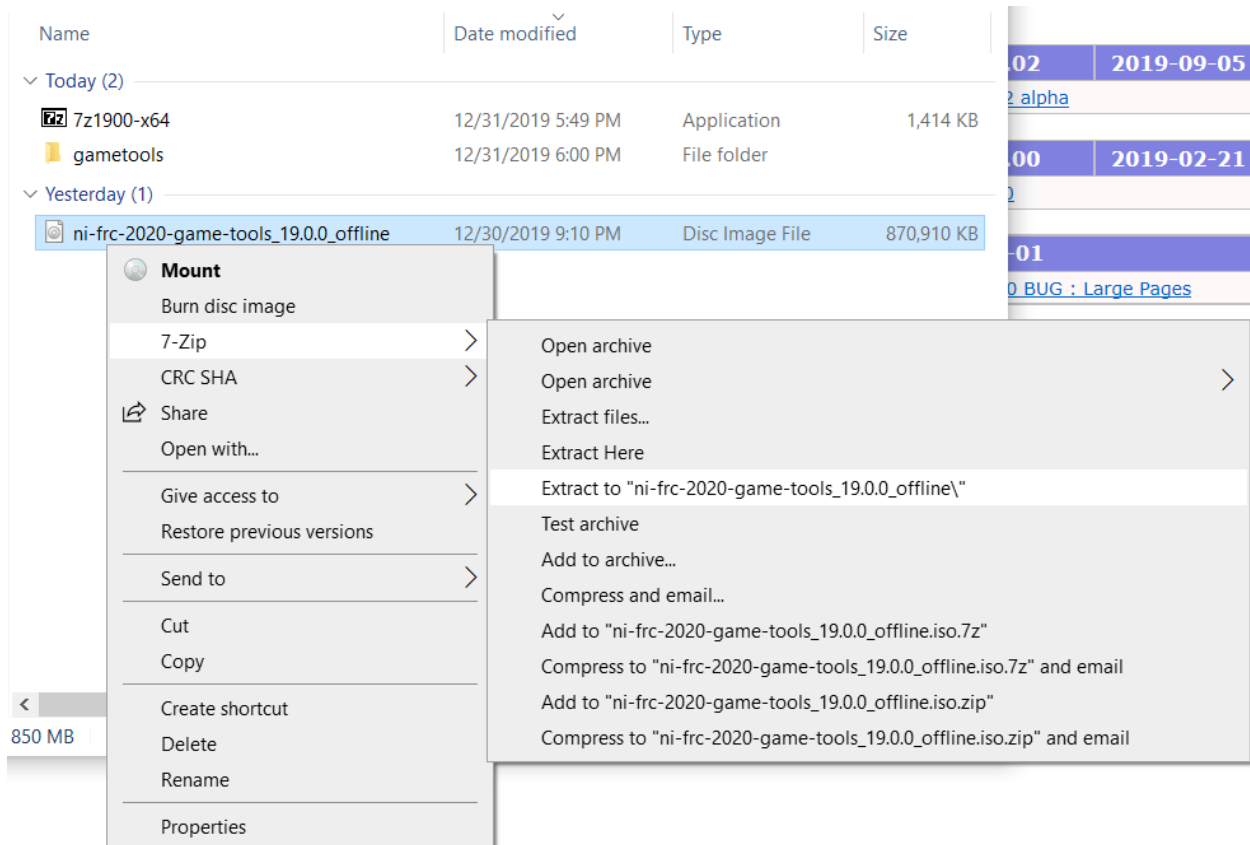
Right click on the downloaded iso file and select mount. Run install.exe from the mounted iso. Click “Yes” if a Windows Security prompt



Note: other installed programs may associate with iso files and the mount option may not appear. If that software does not give the option to mount or extract the iso file, then follow the directions in the “Offline Installer (Windows 7, 8, & 8.1)” tab.

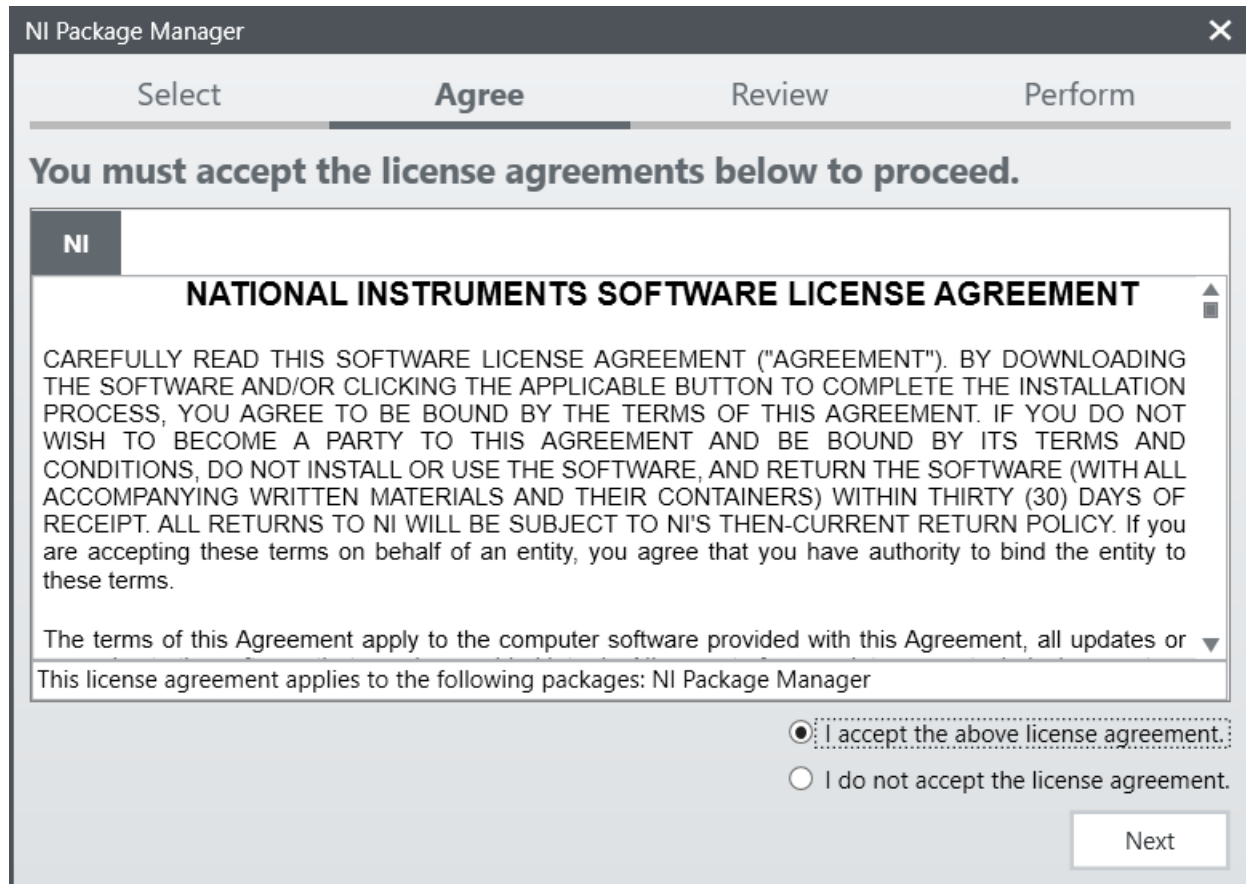
Offline Installer (Windows 7, 8, & 8.1)

Install 7-Zip (download [here](#)). As of the writing of this document, the current released version is 19.00 (2019-02-21). Right click on the downloaded iso file and select Extract to.



Run install.exe from the extracted folder. Click “Yes” if a Windows Security prompt appears. Click “Yes” if a Windows Security prompt appears.

1.6.5 NI Package Manager License



The image shows a screenshot of the 'NI Package Manager' window. At the top, there is a title bar with the text 'NI Package Manager' and a close button. Below the title bar is a progress bar with four steps: 'Select', 'Agree', 'Review', and 'Perform'. The 'Agree' step is currently selected and highlighted. Below the progress bar, a message states: 'You must accept the license agreements below to proceed.' Below this message is a section titled 'NI' with a sub-header 'NATIONAL INSTRUMENTS SOFTWARE LICENSE AGREEMENT'. The main text of the agreement is displayed in a scrollable area. At the bottom of the window, there are two radio buttons: 'I accept the above license agreement.' (which is selected) and 'I do not accept the license agreement.' To the right of these radio buttons is a 'Next' button.

NI Package Manager

Select Agree Review Perform

You must accept the license agreements below to proceed.

NI

NATIONAL INSTRUMENTS SOFTWARE LICENSE AGREEMENT

CAREFULLY READ THIS SOFTWARE LICENSE AGREEMENT ("AGREEMENT"). BY DOWNLOADING THE SOFTWARE AND/OR CLICKING THE APPLICABLE BUTTON TO COMPLETE THE INSTALLATION PROCESS, YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT WISH TO BECOME A PARTY TO THIS AGREEMENT AND BE BOUND BY ITS TERMS AND CONDITIONS, DO NOT INSTALL OR USE THE SOFTWARE, AND RETURN THE SOFTWARE (WITH ALL ACCOMPANYING WRITTEN MATERIALS AND THEIR CONTAINERS) WITHIN THIRTY (30) DAYS OF RECEIPT. ALL RETURNS TO NI WILL BE SUBJECT TO NI'S THEN-CURRENT RETURN POLICY. If you are accepting these terms on behalf of an entity, you agree that you have authority to bind the entity to these terms.

The terms of this Agreement apply to the computer software provided with this Agreement, all updates or

This license agreement applies to the following packages: NI Package Manager

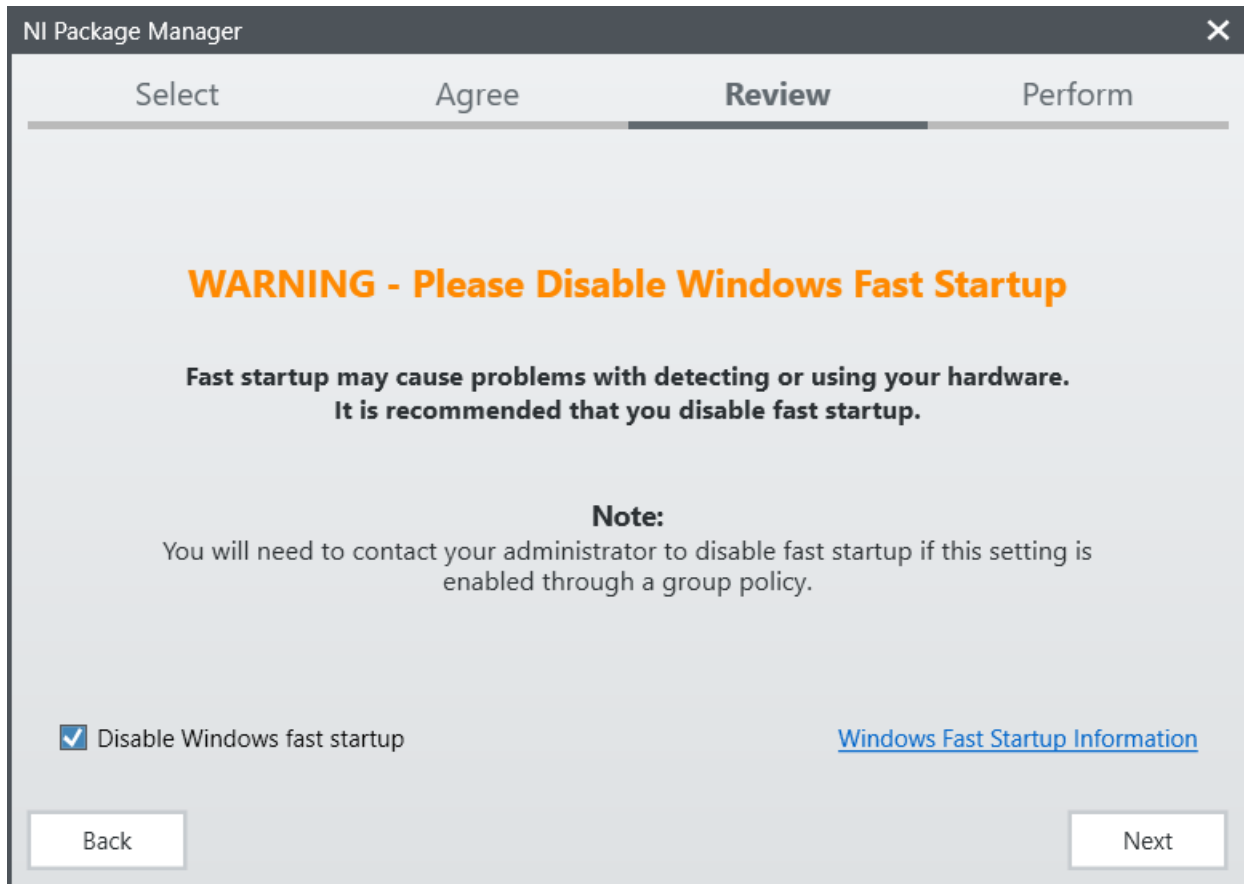
☒ I accept the above license agreement.

☐ I do not accept the license agreement.

Next

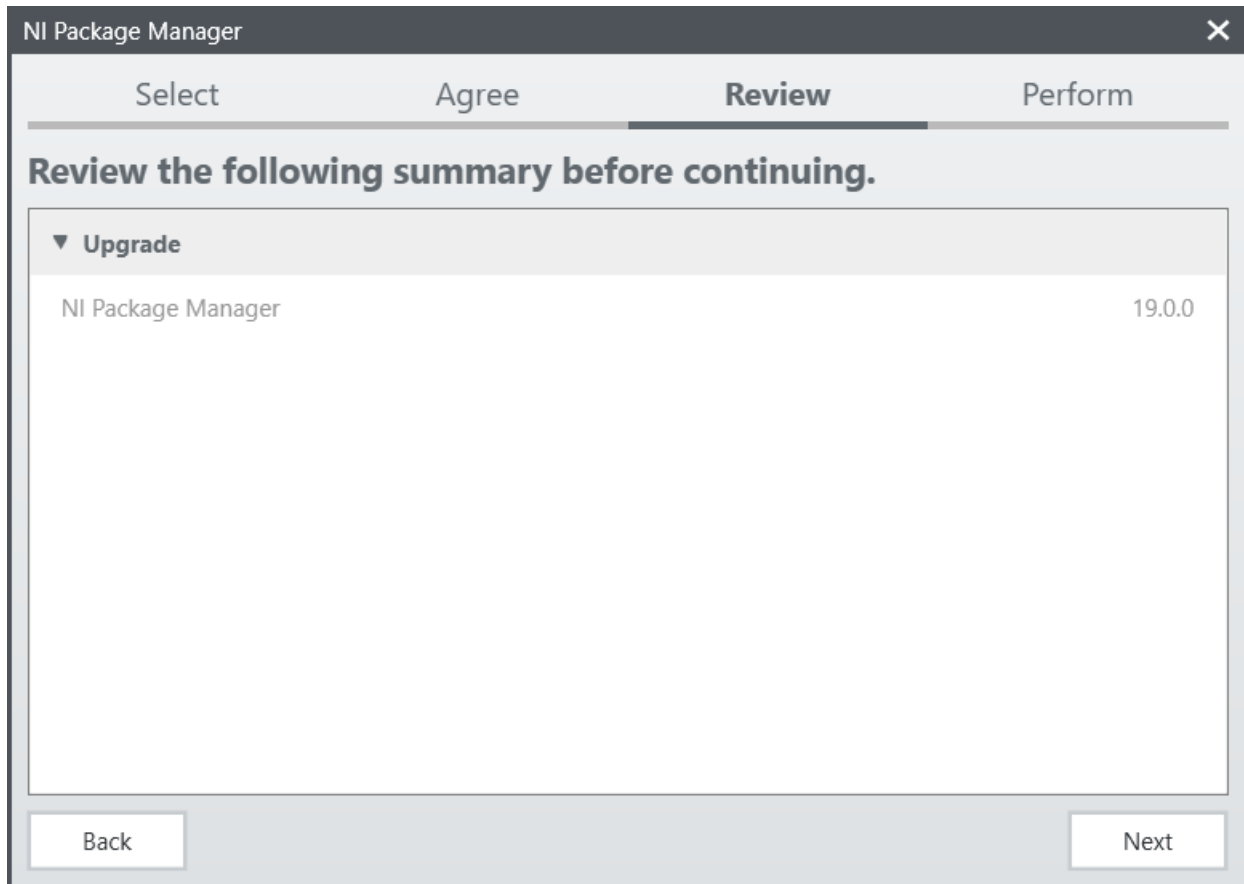
If you see this screen, click "Next"

1.6.6 Disable Windows Fast Startup



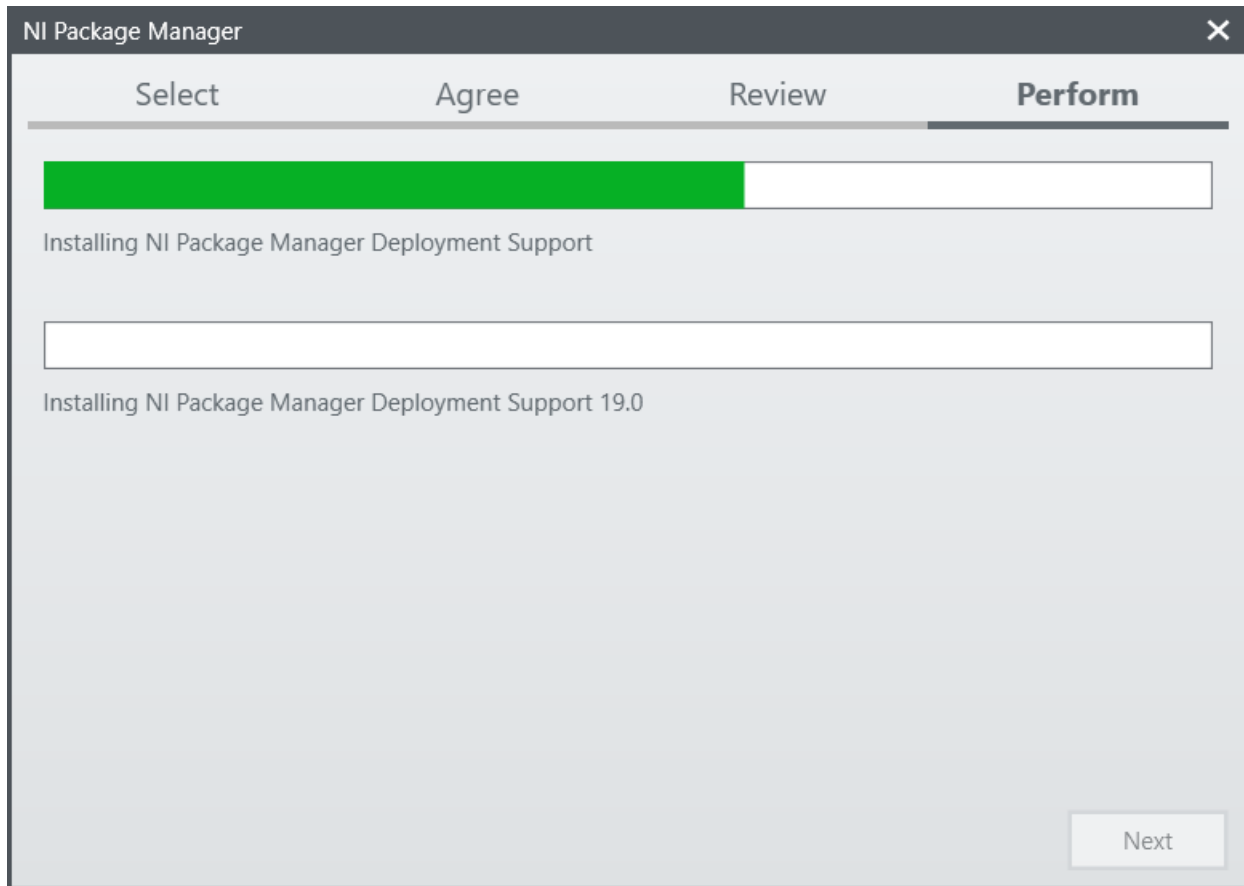
If you see this screen, click “Next”

1.6.7 NI Package Manager Review



If you see this screen, click “Next”

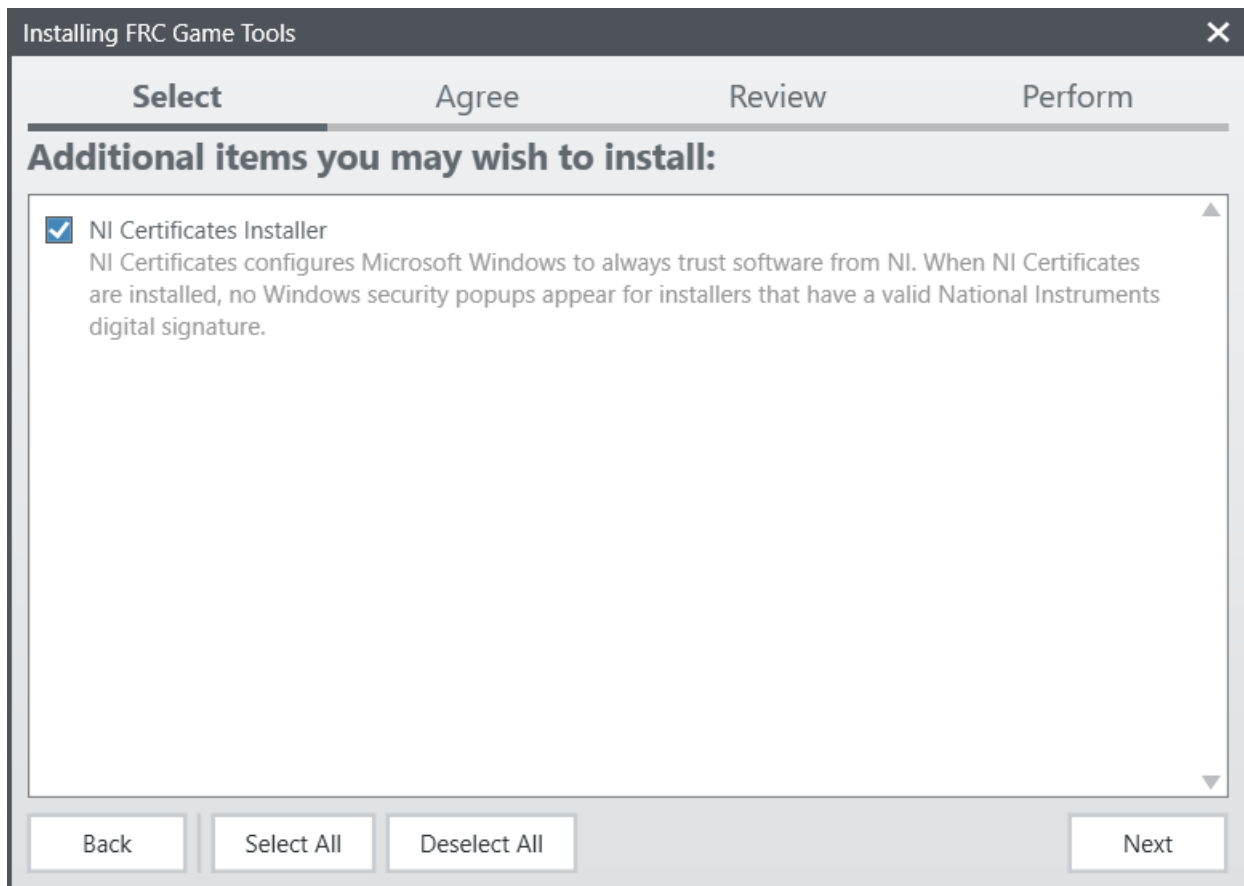
1.6.8 NI Package Manager Installation



Installation progress of the NI Package Manager will be tracked in this window

1.6.9 Product List

1.6.10 Additional Software



If you see this screen, click “Next”

1.6.11 License Agreements

The screenshot shows a window titled "Installing FRC Game Tools" with a close button (X) in the top right corner. The window has a progress bar with four steps: "Select", "Agree" (which is highlighted), "Review", and "Perform". Below the progress bar, a message states: "You must accept the license agreements below to proceed." Below this message is a section for "NI FIRST Competition". The main content area displays the "NATIONAL INSTRUMENTS SOFTWARE LICENSE AGREEMENT". The text of the agreement is as follows: "CAREFULLY READ THIS SOFTWARE LICENSE AGREEMENT ('AGREEMENT'). BY DOWNLOADING THE SOFTWARE AND/OR CLICKING THE APPLICABLE BUTTON TO COMPLETE THE INSTALLATION PROCESS, YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT WISH TO BECOME A PARTY TO THIS AGREEMENT AND BE BOUND BY ITS TERMS AND CONDITIONS, DO NOT INSTALL OR USE THE SOFTWARE, AND RETURN THE SOFTWARE (WITH ALL ACCOMPANYING WRITTEN MATERIALS AND THEIR CONTAINERS) WITHIN THIRTY (30) DAYS OF RECEIPT. ALL RETURNS TO NI WILL BE SUBJECT TO NI'S THEN-CURRENT RETURN POLICY. If you are accepting these terms on behalf of an entity, you agree that you have authority to bind the entity to these terms." Below this text, it says: "The terms of this Agreement apply to the computer software provided with this Agreement, all updates or upgrades to the software that may be provided later by NI as part of any maintenance, technical support, or". This line is underlined. Below the underlined text, it says: "This license agreement applies to the following packages: NI FIRST Robotics Utilities". At the bottom right of the window, there are two radio buttons: "I accept the above 2 license agreements." (which is selected) and "I do not accept all the license agreements." At the bottom left, there is a "Back" button, and at the bottom right, there is a "Next" button.

Installing FRC Game Tools

Select Agree Review Perform

You must accept the license agreements below to proceed.

NI FIRST Competition

NATIONAL INSTRUMENTS SOFTWARE LICENSE AGREEMENT

CAREFULLY READ THIS SOFTWARE LICENSE AGREEMENT ("AGREEMENT"). BY DOWNLOADING THE SOFTWARE AND/OR CLICKING THE APPLICABLE BUTTON TO COMPLETE THE INSTALLATION PROCESS, YOU AGREE TO BE BOUND BY THE TERMS OF THIS AGREEMENT. IF YOU DO NOT WISH TO BECOME A PARTY TO THIS AGREEMENT AND BE BOUND BY ITS TERMS AND CONDITIONS, DO NOT INSTALL OR USE THE SOFTWARE, AND RETURN THE SOFTWARE (WITH ALL ACCOMPANYING WRITTEN MATERIALS AND THEIR CONTAINERS) WITHIN THIRTY (30) DAYS OF RECEIPT. ALL RETURNS TO NI WILL BE SUBJECT TO NI'S THEN-CURRENT RETURN POLICY. If you are accepting these terms on behalf of an entity, you agree that you have authority to bind the entity to these terms.

The terms of this Agreement apply to the computer software provided with this Agreement, all updates or upgrades to the software that may be provided later by NI as part of any maintenance, technical support, or

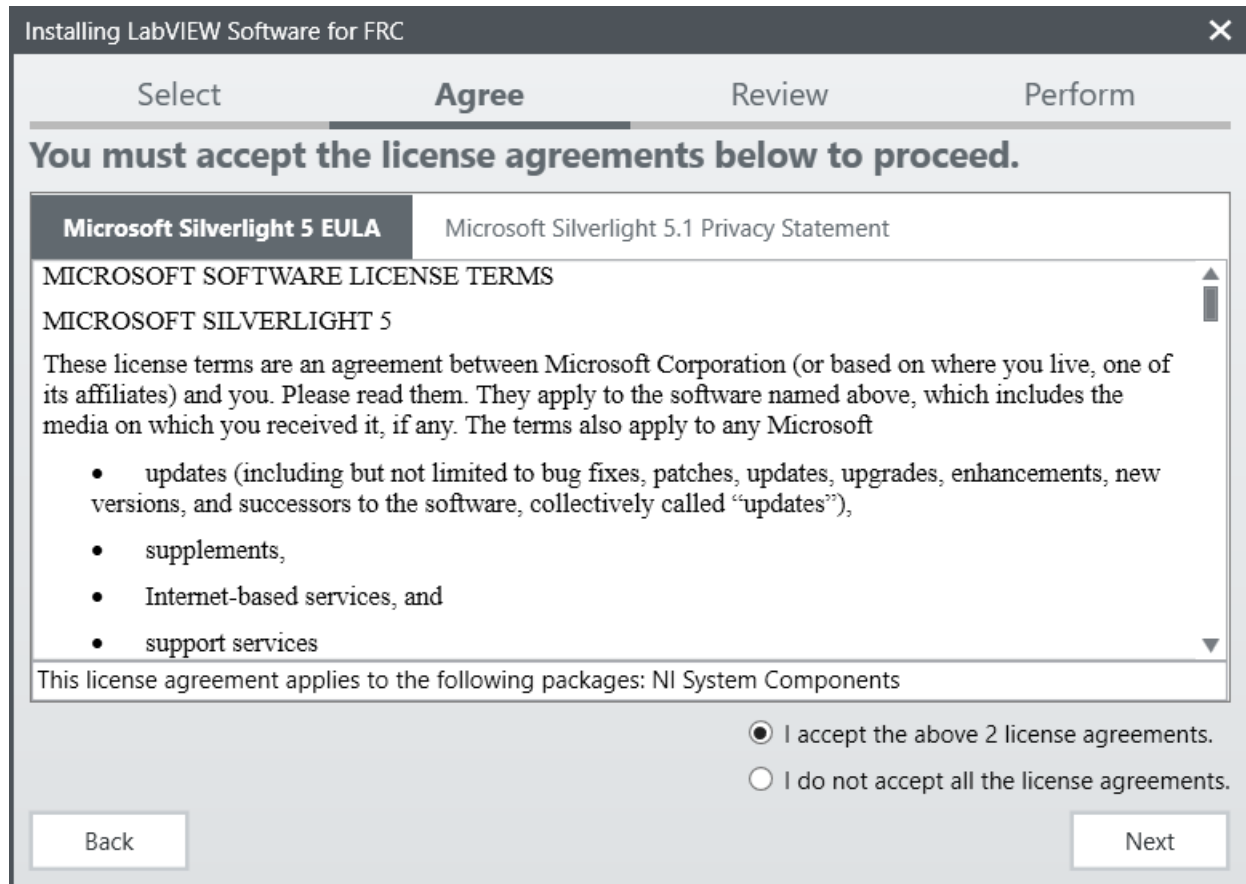
This license agreement applies to the following packages: NI FIRST Robotics Utilities

☒ I accept the above 2 license agreements.
☐ I do not accept all the license agreements.

Back Next

Select "I accept..." then click "Next"

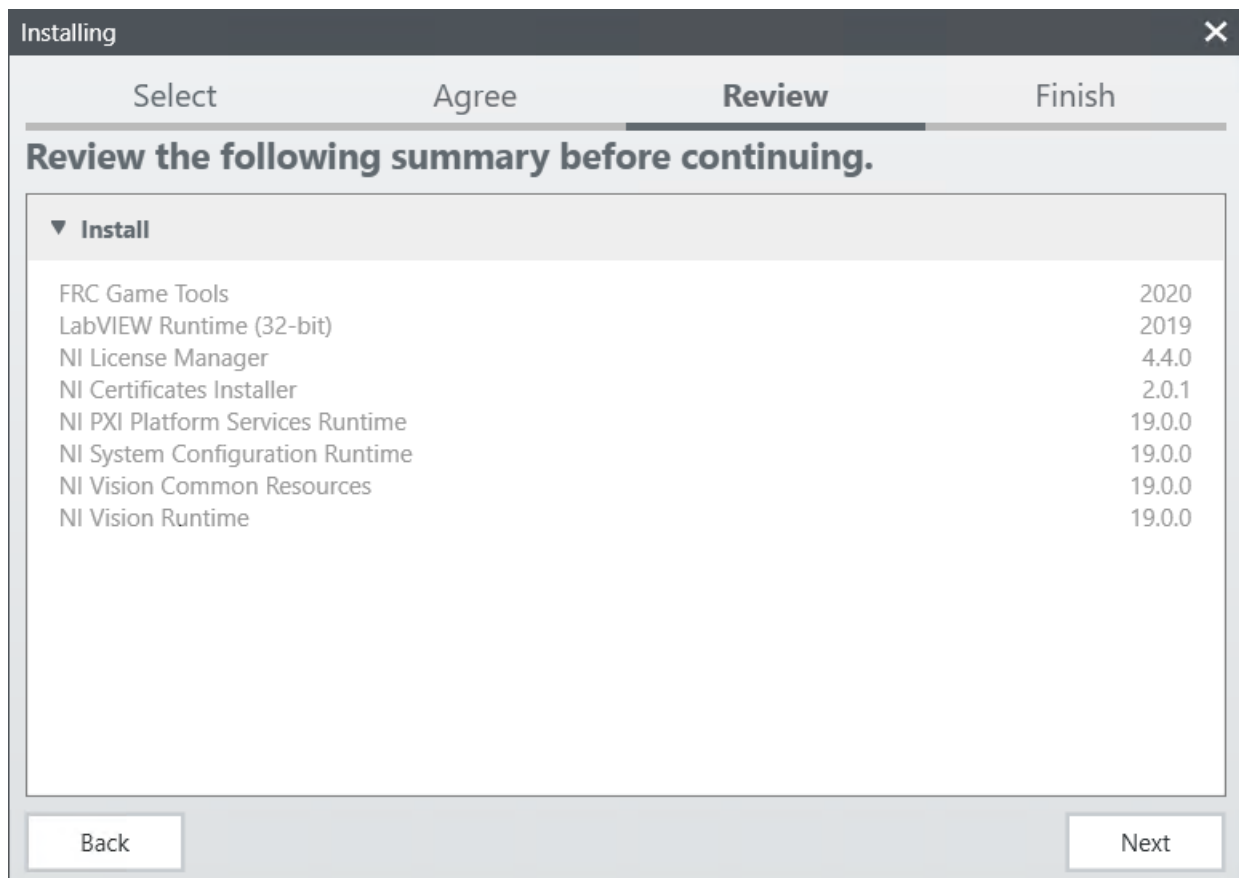
1.6.12 License Agreements Page 2



If you see this screen, select “I accept...” then click “Next”

1.6.13 Review Summary

Click Next



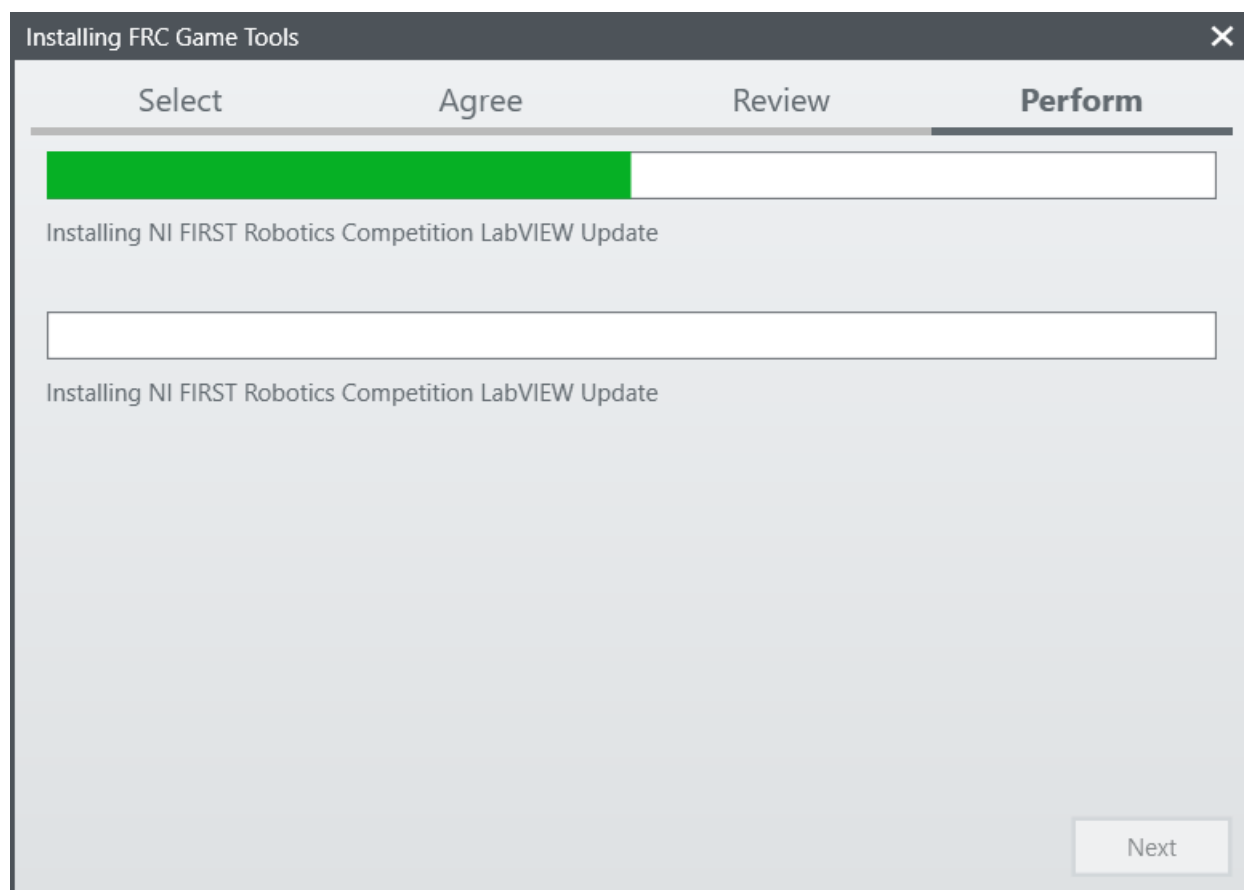
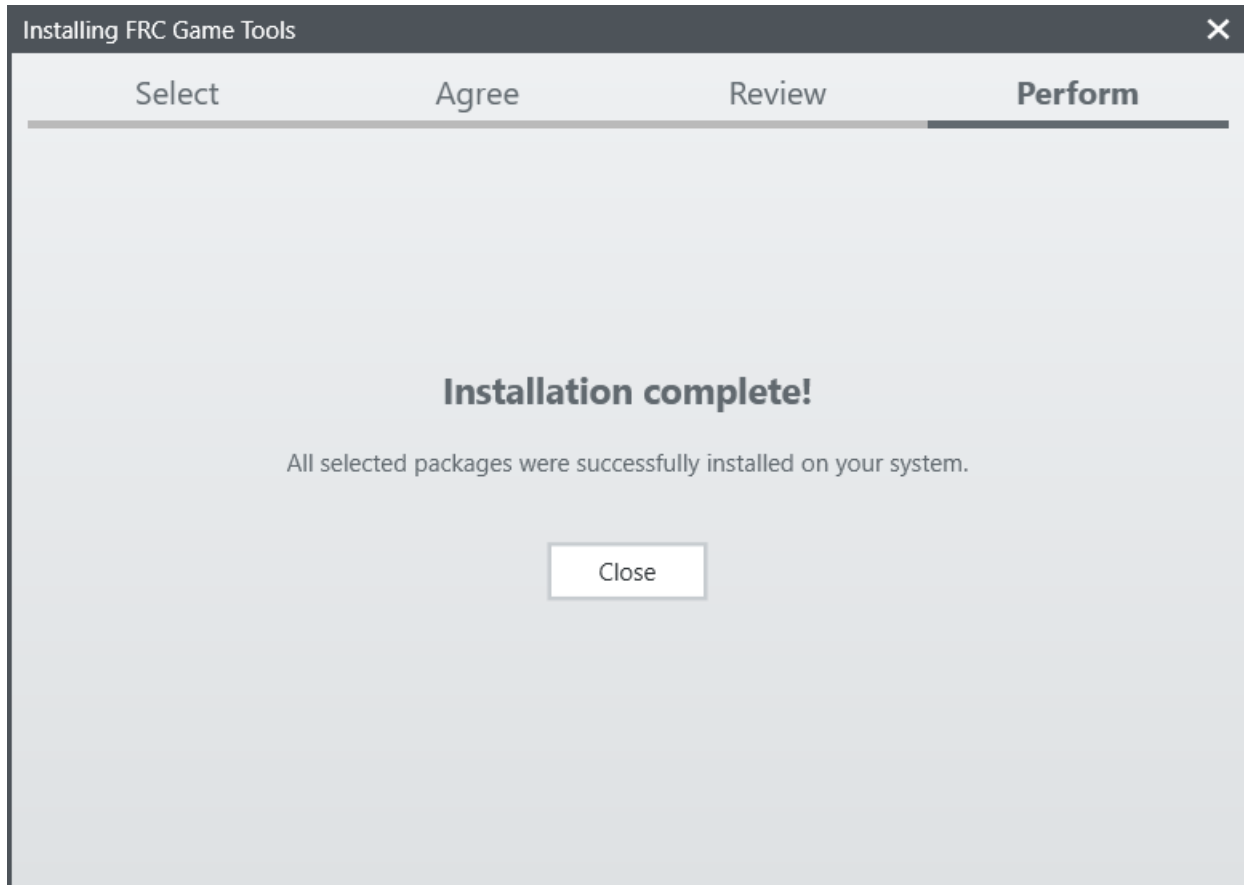


Fig. 1: Detail Progress

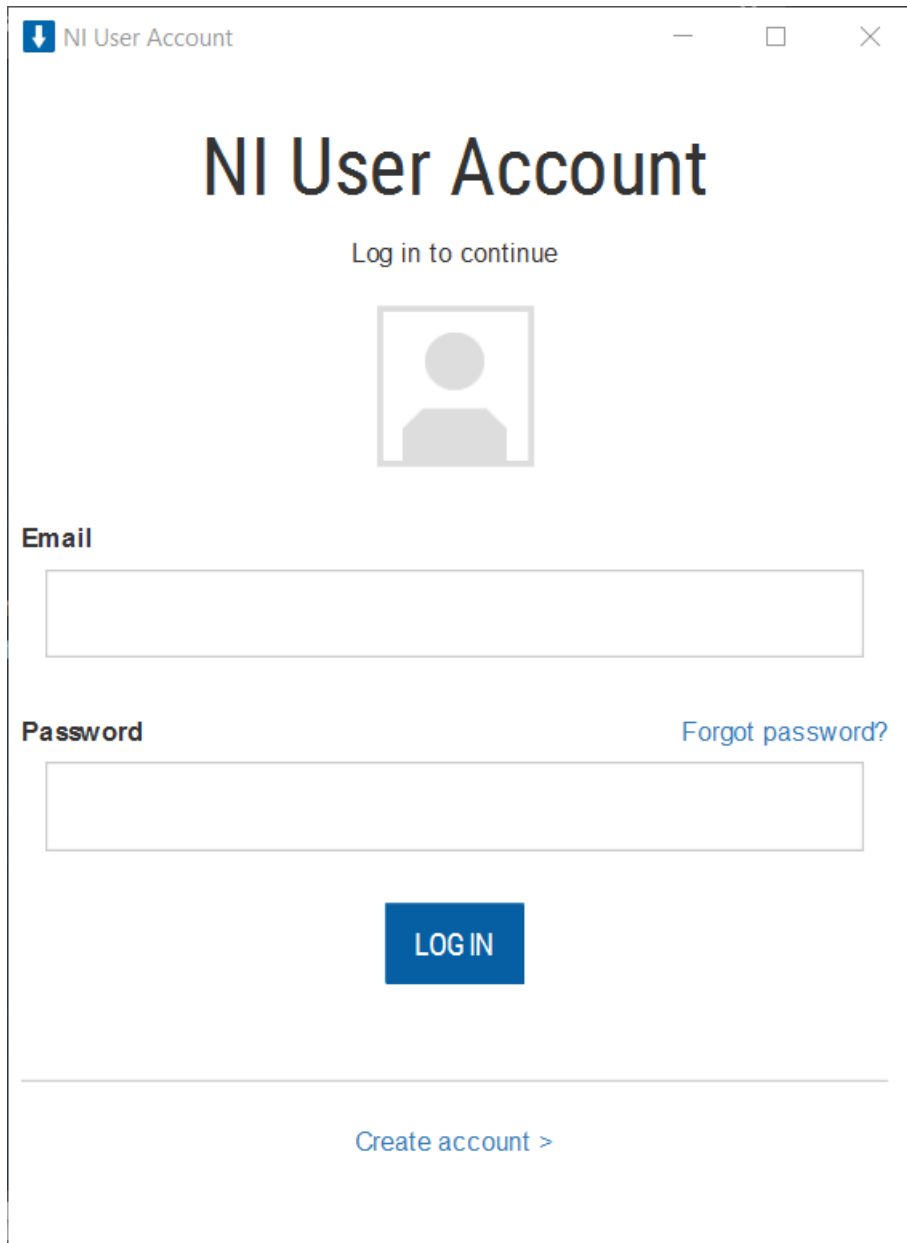
1.6.14 Detail Progress

1.6.15 Installation Summary



click Close

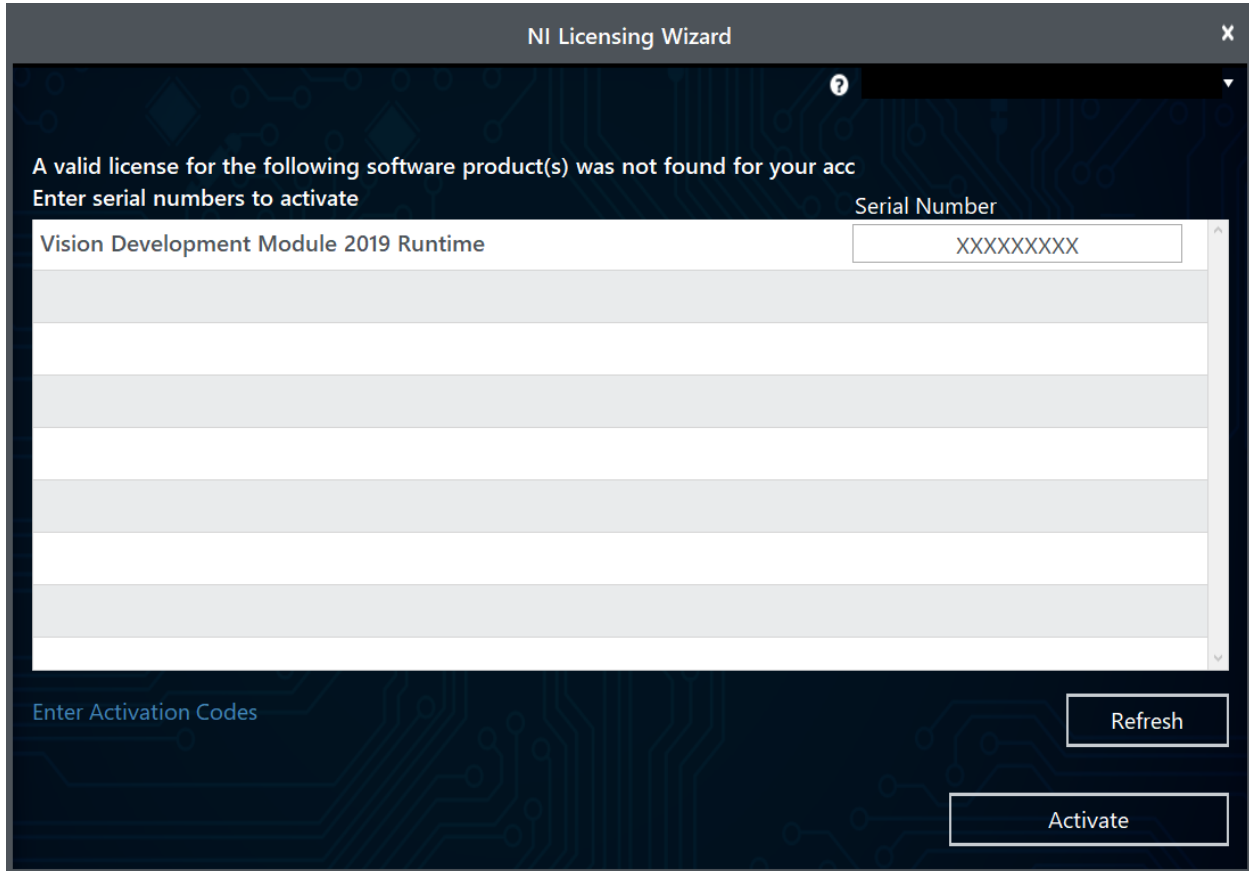
1.6.16 NI Activation Wizard



The screenshot shows a window titled "NI User Account" with standard window controls (minimize, maximize, close). The main heading is "NI User Account" in a large, bold font. Below it is the text "Log in to continue". There is a placeholder for a user profile picture. Below the picture are two input fields: "Email" and "Password". To the right of the "Password" field is a link "Forgot password?". Below the input fields is a blue "LOG IN" button. At the bottom of the window, separated by a horizontal line, is a link "Create account >".

Log into your ni.com account. If you don't have an account, select 'Create account' to create a free account.

1.6.17 NI Activation Wizard (2)

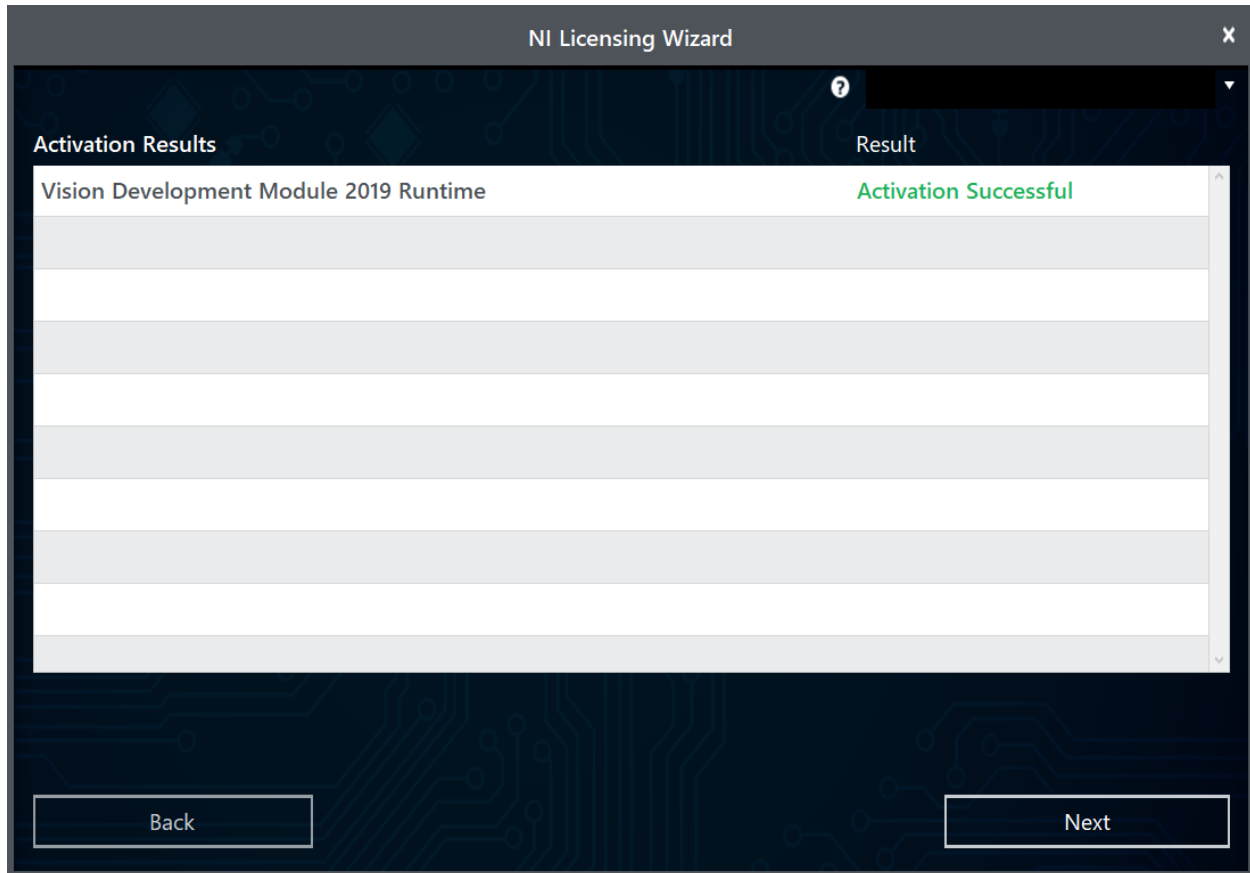


The image shows a screenshot of the "NI Licensing Wizard" window. The window has a dark blue background with a circuit pattern. At the top, the title bar says "NI Licensing Wizard" with a close button (X) on the right. Below the title bar, there is a message: "A valid license for the following software product(s) was not found for your acc" and "Enter serial numbers to activate". To the right of this message is a "Serial Number" label. Below the message is a table with two columns: "Software Product" and "Serial Number". The first row of the table contains "Vision Development Module 2019 Runtime" and "XXXXXXXXXX". There are several empty rows below it. At the bottom left of the window, there is a link that says "Enter Activation Codes". At the bottom right, there are two buttons: "Refresh" and "Activate".

Software Product	Serial Number
Vision Development Module 2019 Runtime	XXXXXXXXXX

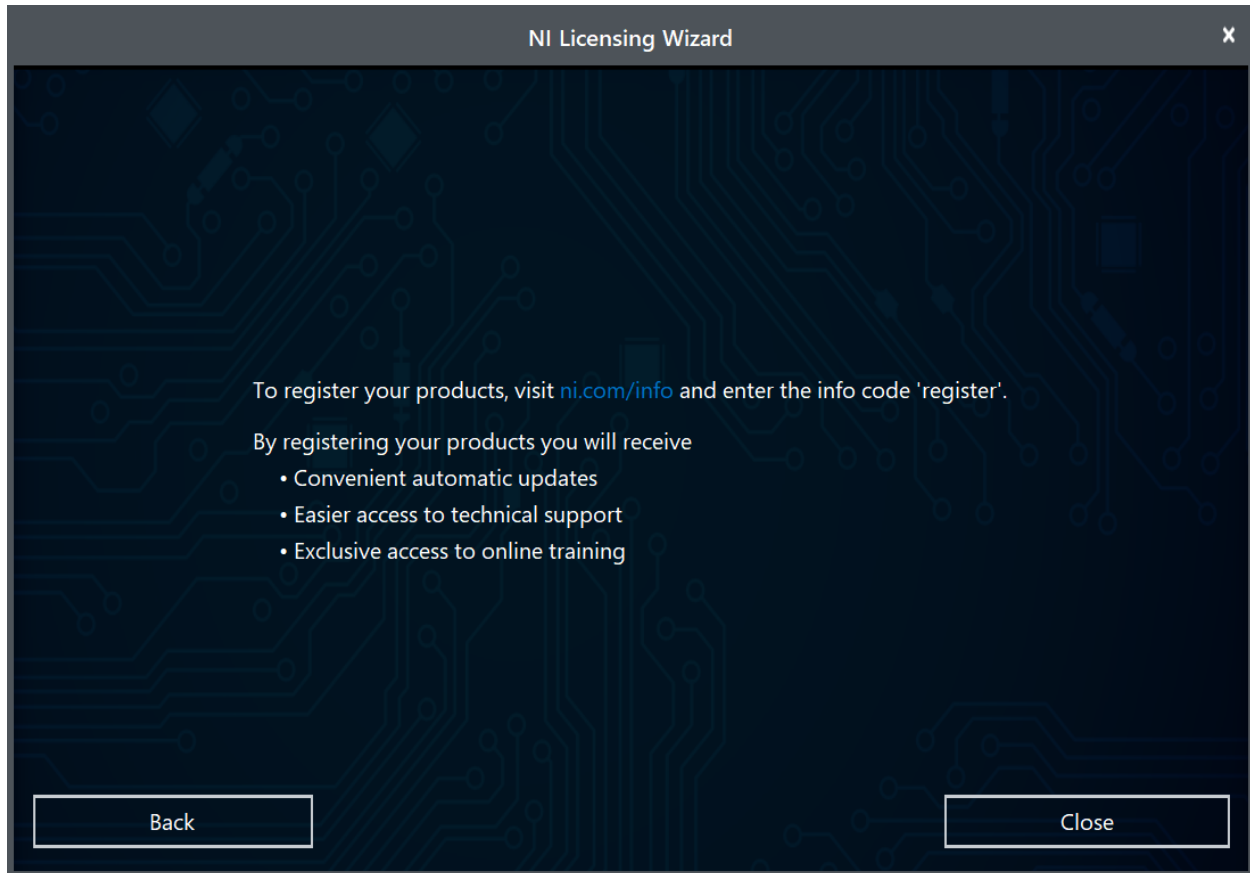
Enter the serial number. Click "Activate". Note: If this is the first time activating this year's software on this account, you will see the message shown above about a valid license not being found. You can ignore this.

1.6.18 NI Activation Wizard (3)



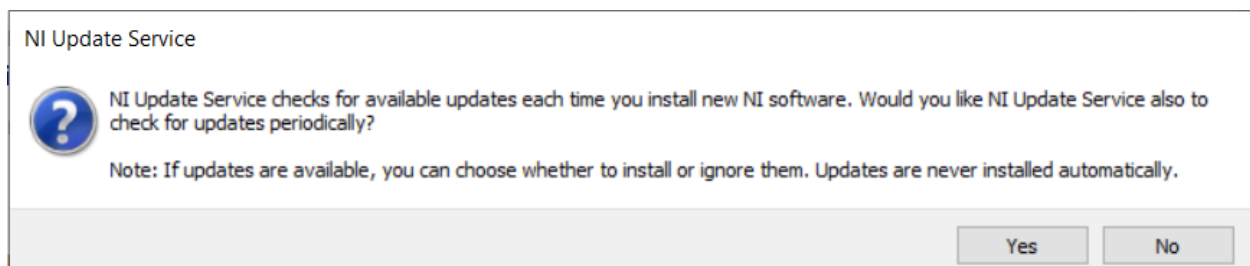
If your products activate successfully, an “Activation Successful” message will appear. If the serial number was incorrect, it will give you a text box and you can re-enter the number and select “Try Again”. If everything activated successfully, click “Next”.

1.6.19 NI Activation Wizard (4)



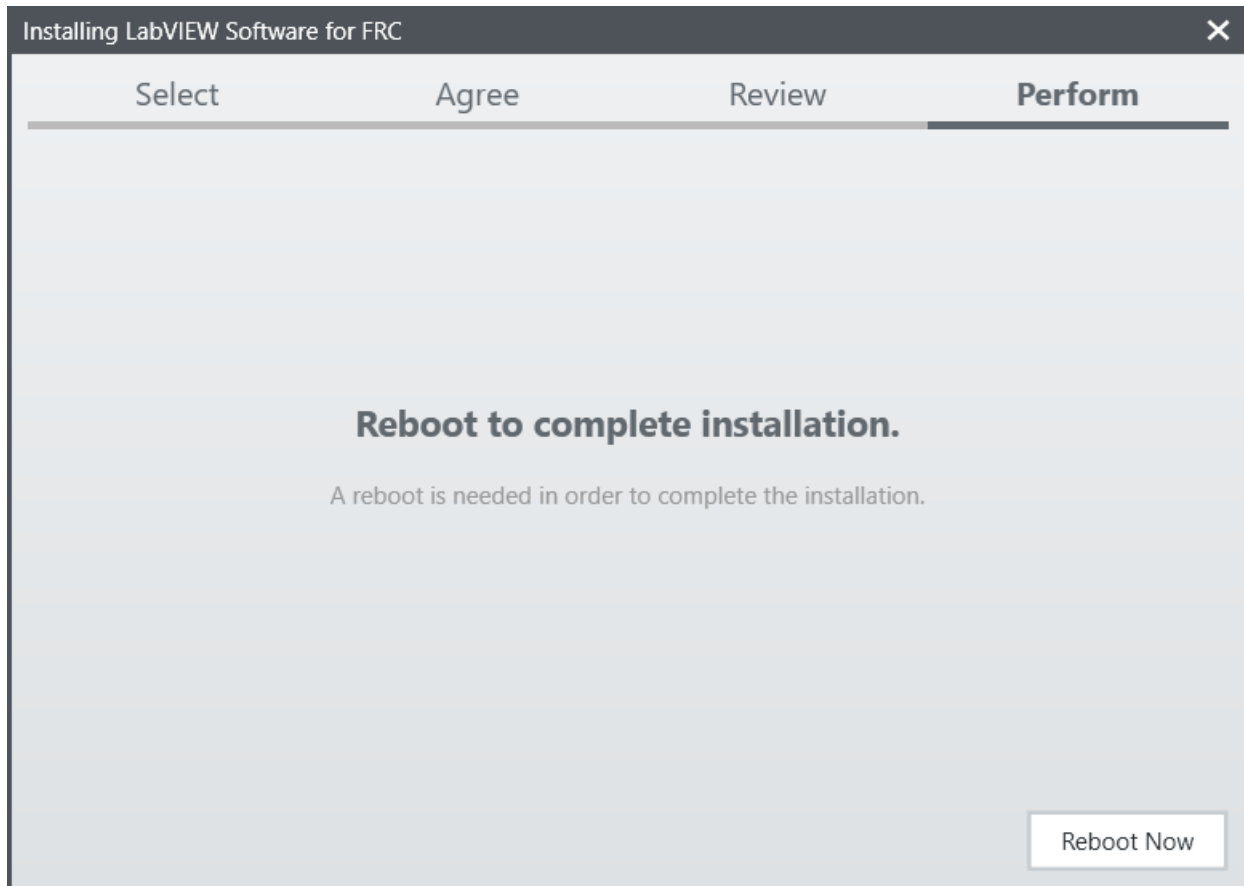
Click "Close".

1.6.20 NI Update Service



You will be prompted whether to enable the NI update service. You can choose to not enable the update service.

1.6.21 Reboot to Complete Installation



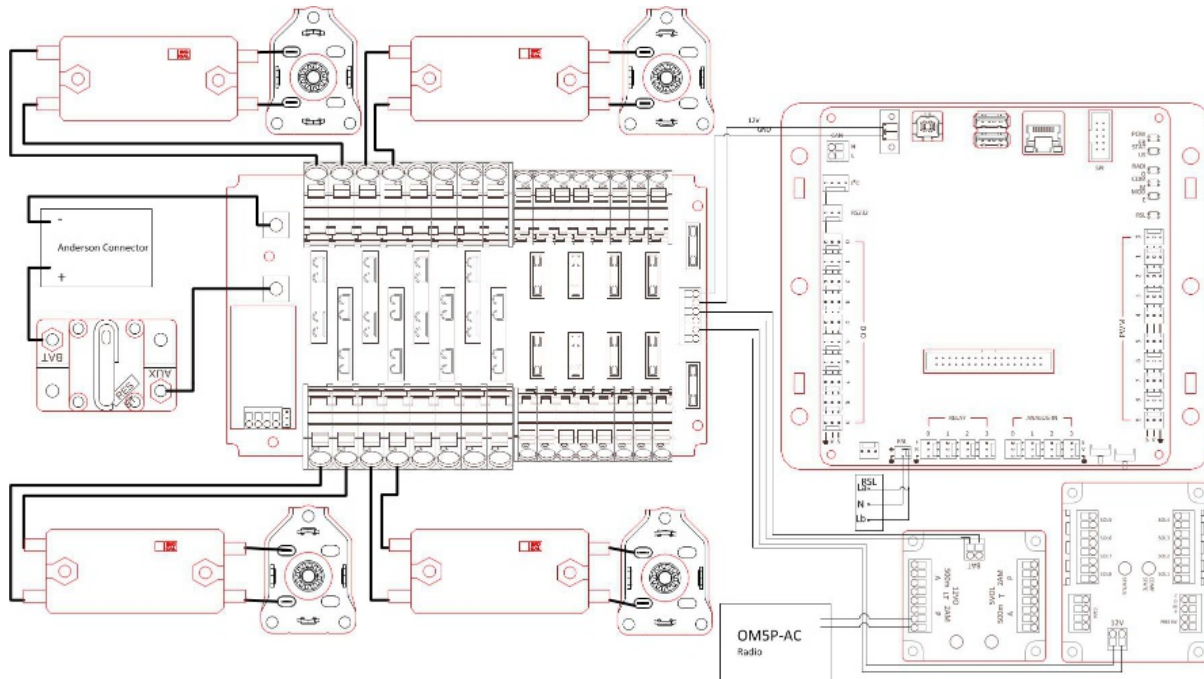
If prompted, Select “Reboot Now” after closing any open programs.

1.7 Como fazer o cabeamento de um robo para FRC

Note: Este documento descreve como fazer o cabeamento de uma chapa com componentes eletrônicos para testes.

Algumas das imagens presentes nessa seção demonstram o Setup do sistema de controle de um robô que utiliza controladores de motor Spark. Os diagramas e layouts do cabeamento a seguir são bem parecidos com o dos demais controladores. Há também um segundo set de imagens que demonstram o passo a passo do cabeamento de controladores PWM com fios integrados.

1.7.1 Materiais e componentes



Você vai precisar dos seguintes componentes eletrônicos e ferramentas

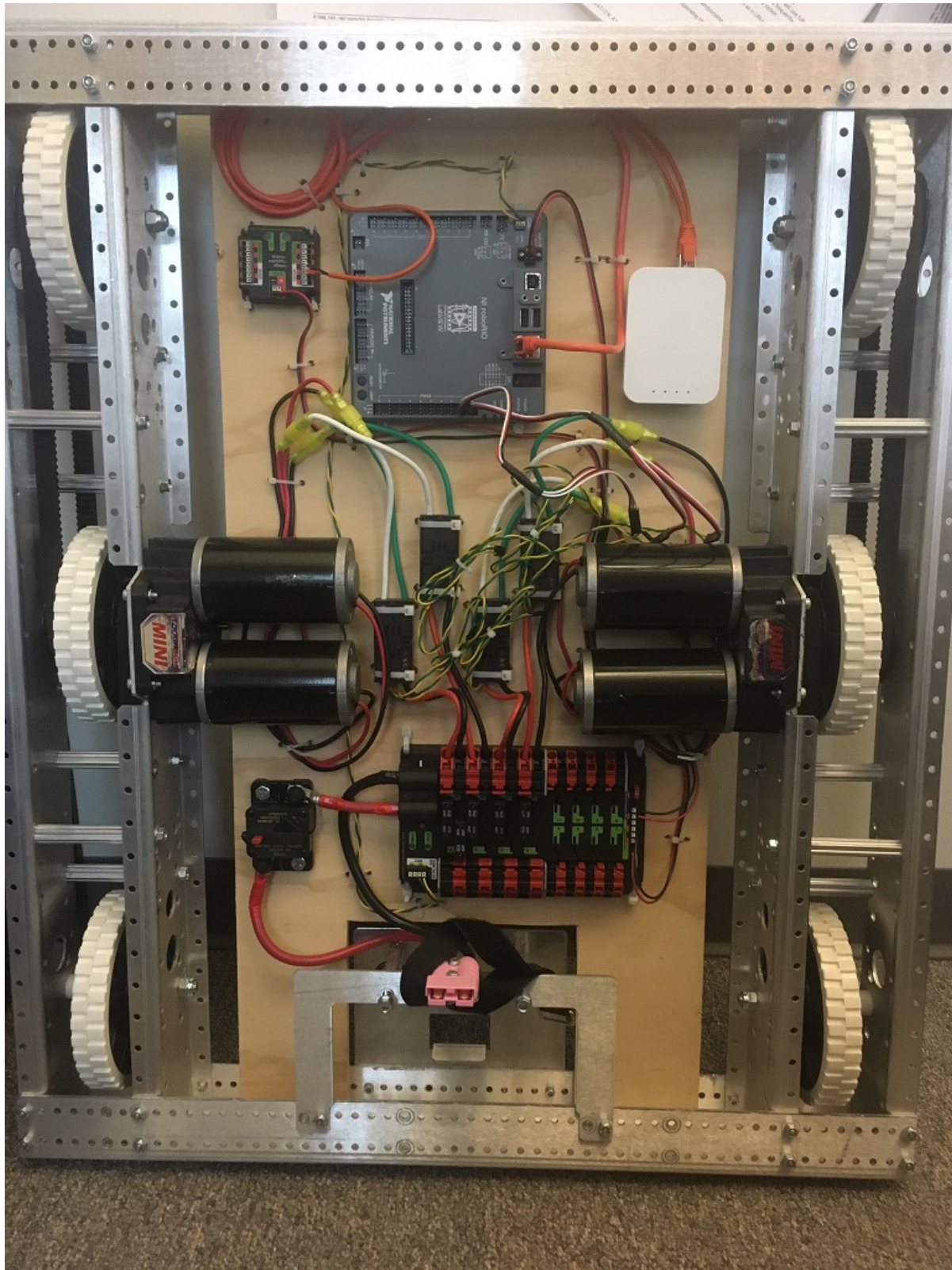
- Materiais do KIT:
 - Power Distribution Panel (PDP)
 - roboRIO
 - Pneumatics Control Module (PCM)
 - Voltage Regulator Module (VRM)
 - OpenMesh radio (com seu cabo de alimentação POE)
 - Robot Signal Light (RSL)
 - 4x Victor SPX ou outro controlador de motor
 - 120A Circuit breaker
 - 4x 40A Circuit breaker
 - Fio vermelho e preto de 6 AWG (4,1mm de seção)
 - Fio vermelho e preto de 10 AWG (2,5mm de seção)
 - Fio vermelho e preto de 18 AWG (1mm de seção)
 - Fio amarelo e verde de 22AWG
 - 16x 10-12 AWG terminais em olho (amarelos)
 - 2x Anderson SB50
 - Terminais em olho de 6 AWG
 - Bateria 12V
 - Fita isolante

- Zip ties
- Compensado de 1/2" ou 1/4". Ou Policarbonato
- Ferramentas necessárias:
 - Chave de fenda pequena
 - Chave de fenda muito pequena (do tipo utilizado em ajuste de óculos)
 - Chave Phillips
 - Chave Allen de 5mm
 - Chave Allen de 1/16"
 - Alicates desmontador e de corte
 - Chave de boca de 7/16"

1.7.2 Construindo a base do para o Sistema de Controle

Para a produção da base, corte uma chapa (madeira ou plástico) de 1/4" ou 1/2" de espessura e aproximadamente 24" x 16". Se for o caso de um chassi pré-fabricado, consulte a documentação e verifique o tamanho ideal para a configuração do chassi correspondente.

1.7.3 Organize os principais componentes do Sistema de Controle



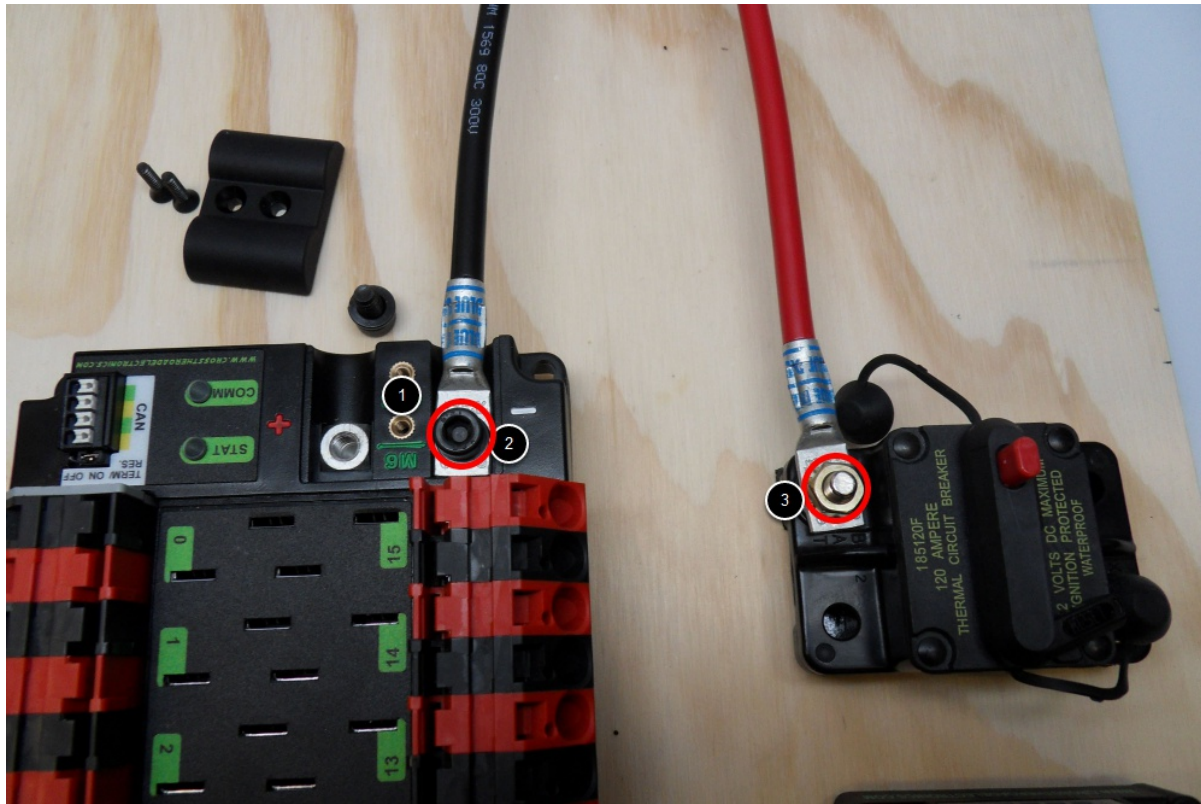
Organize os componentes na chapa. O layout da imagem acima é um bom exemplo.



1.7.4 Fixe os componentes

Utilizando fitas dupla-face (VHB por exemplo) ou ZIP ties, prenda todos componentes à base. Tenha em mente que, em muitos dos jogos da FRC, há constante toque físico entre os robôs. Muitos times optam por utilizar fitas dupla face e, principalmente, ZIP ties para garantir a fixação apropriada dos componentes.

1.7.5 Fixe o conector da bateria à PDP

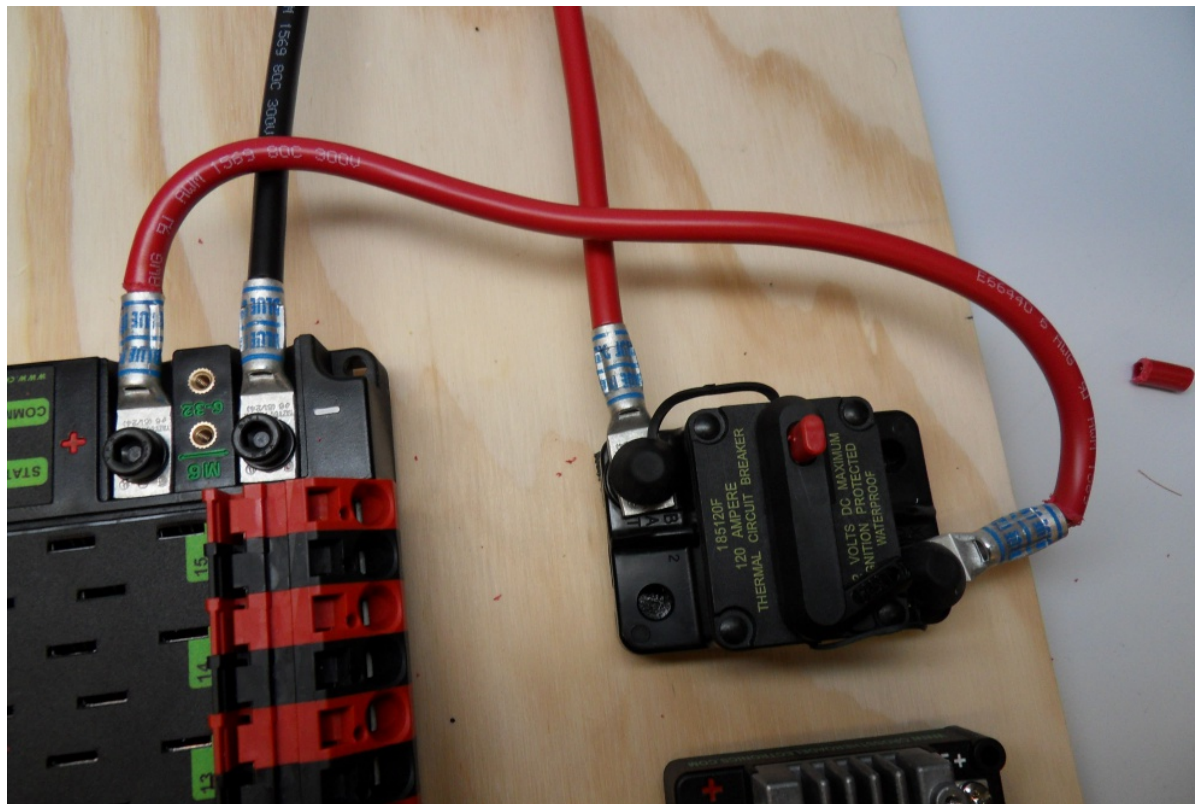


Requer: Conector Anderson, Terminal olho de 6AWG , Allen de 1/16", Alle de 5mm e a chave de boca 7/16"

Prenda os terminais em olho ao Conector da bateria:

1. Utilizando a Allen 1/16", retire os dois parafusos que prendem a proteção dos terminais da PDP.
2. Utilizando a Allen 5mm, remova o parafuso e a arruela que fica rosqueadas ao o polo negativo da PDP e prenda o terminal negativo do conector da bateria.
3. Utilizando a chave de boca 7/16", remova a proteção de borracha e a porca do parafuso do Disjuntor principal, encaixe o terminal positivo do conector de bateria e prenda-o bem com a porca.

1.7.6 Conecte o Disjuntor principal à PDP

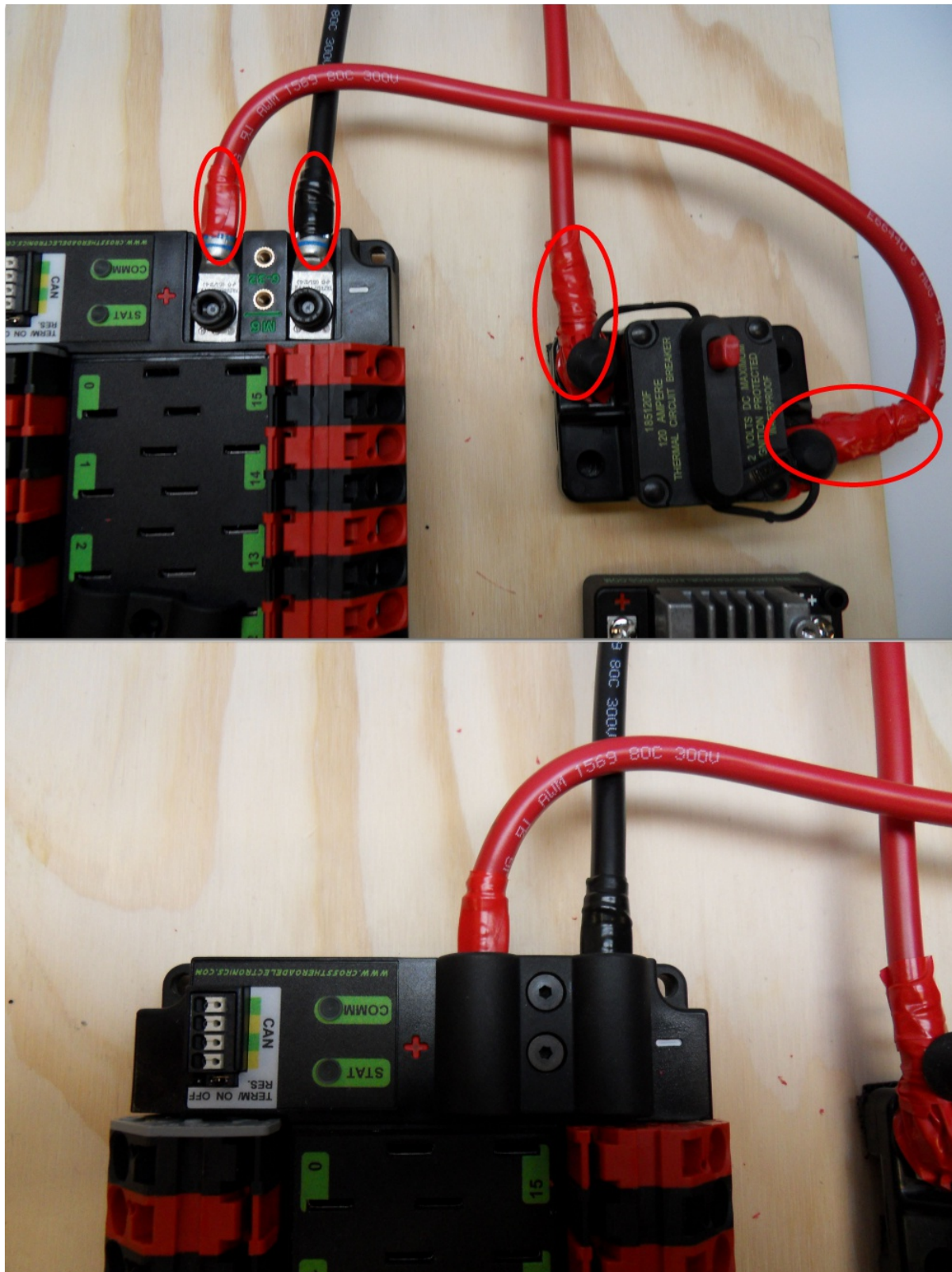


Requer: Fio vermelho de 6AWG, 2x terminais olho de 6AWG, Allen de 5mm e a chave de boca 7/16".

Crimpe um terminal olho na ponta do fio vermelho de 6AWG. Retire a porca rosqueada ao parafuso do lado "AUX" do disjuntor principal (utilizando a chave de boca 7/16"), encaixe o terminal do cabo no parafuso e rosqueie a porca, prendendo-o. Meça o comprimento necessário para que o cabo chegue ao terminal positivo da PDP.

1. Corte, encaixa e crimpe o terminal ao outro lado do fio vermelho de 6AWG.
2. Utilizando a chave de boca 7/16", prenda o fio ao lado "AUX" do disjuntor principal de 120A.
3. Utilizando a Allen de 5mm, prenda a outra ponta ao terminal positivo da PDP.

1.7.7 Isole as conexões da PDP



Requer: Allen 1/16", Fita isolante

1. Utilizando a fita isolante, isole as duas conexões do disjuntor principal. Isole a parte dos terminais da PDP que irão entrar em contato com a proteção quando for presa novamente. Uma maneira de isolar as conexões do disjuntor principal é passar a fita no cabo e na porca antes de serem presos e, depois de presos, passar fita novamente.
2. Utilizando a Allen de 1/16", prenda a proteção dos terminais à PDP.

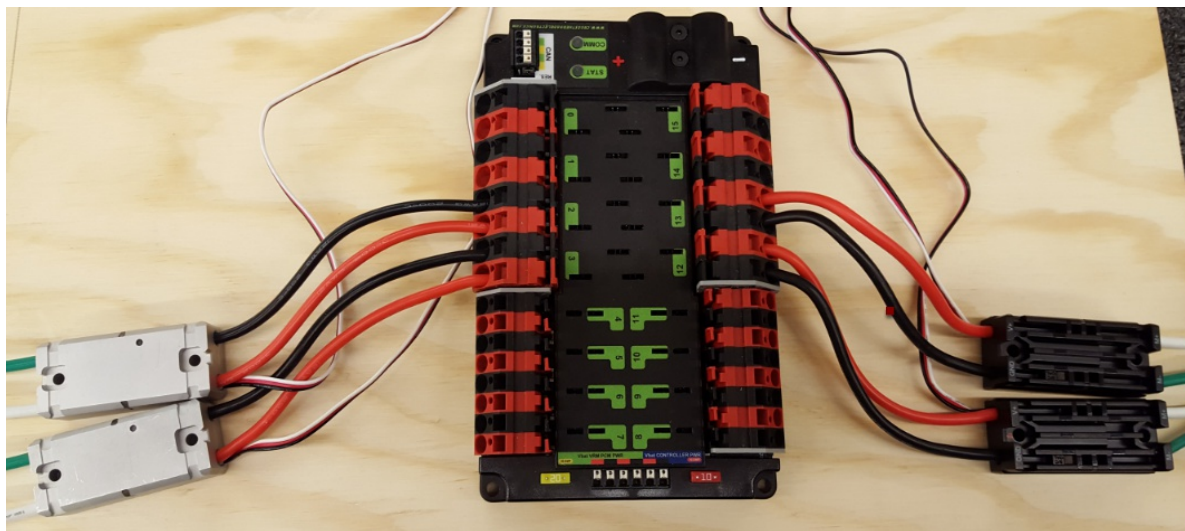
1.7.8 Conectores Wago

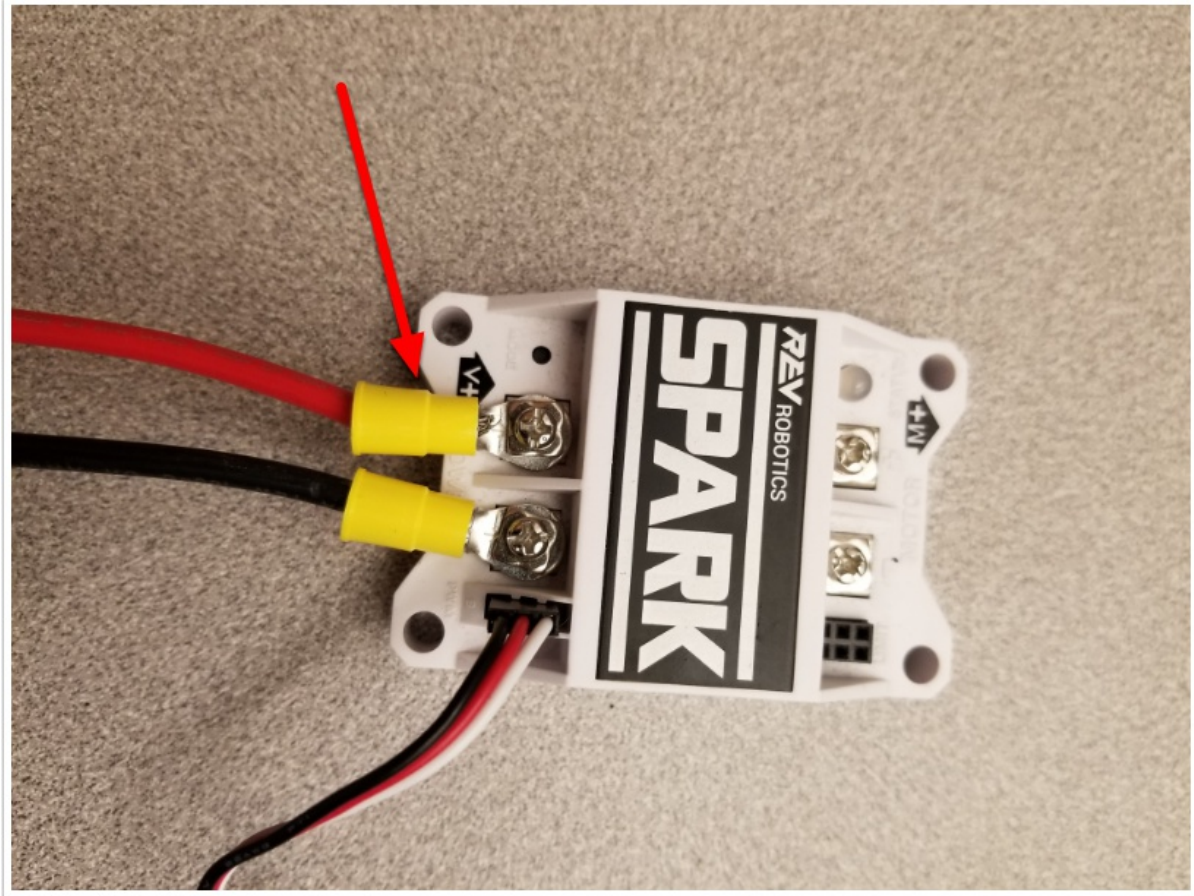
O próximo passo é aprender a utilizar os conectores *Wago* na PDP. Para utiliza-los, pegue uma pequena chave de fenda e a insira no buraco retangular (na lateral da PDP) o mais horizontal possível. Empurre a chave para cima, aí você vai poder ver que, abaixo do buraco retangular, uma pequena entrada vai se abrir. Nessa entrada devem ser presos os cabos dos componentes a serem energizados. A PDP têm dois tipos de conectores:

- Conector *Wago* pequeno: Aceita 10AWG-24AWG, strip 11-12mm (~7/16")
- Conector *Wago* grande: Aceita 6AWG-12AWG, strip 12-13mm(~1/2")

Para facilitar a entrada do cabo na entrada da PDP, deve deixar a parte de cobre exposta bem esticada (não torcida, de preferencia).

1.7.9 Energizando controladores de motor





Requer: Alicates para desencapar fios, Chave de fenda pequena, fio de 10 ou 12 AWG, terminal olho de 10 ou 12, alicate crimpador

Para o Victor SPX ou outro controlador de motor com fios integrados (imagem de cima): - Desencape a ponta dos cabos de energia (vermelho e preto) e insira na entrada (conector Wago) de 40 amperes da PDP.

Para controladores de motor com terminais (imagem de baixo)

1. Corte um cabo vermelho e um cabo preto do tamanho apropriado para chegar até a entrada de 40 amperes da PDP (é recomendado que seja um pouco maior do que o necessário);
2. Insira a parte desencapada dos fios na PDP conforme a polaridade correspondente;
3. Crimpe um terminal olho na outra ponta do cabo;
4. Prenda os fios nos terminais dos controladores (vermelho no + e preto no -).

1.7.10 Conector Weidmuller

O tamanho correto para se desencapar é 5/16" (~8mm), e não 5/8" como dito no vídeo.

Para utilizar estes conectores de maneira mais eficiente, tenha em mente que:

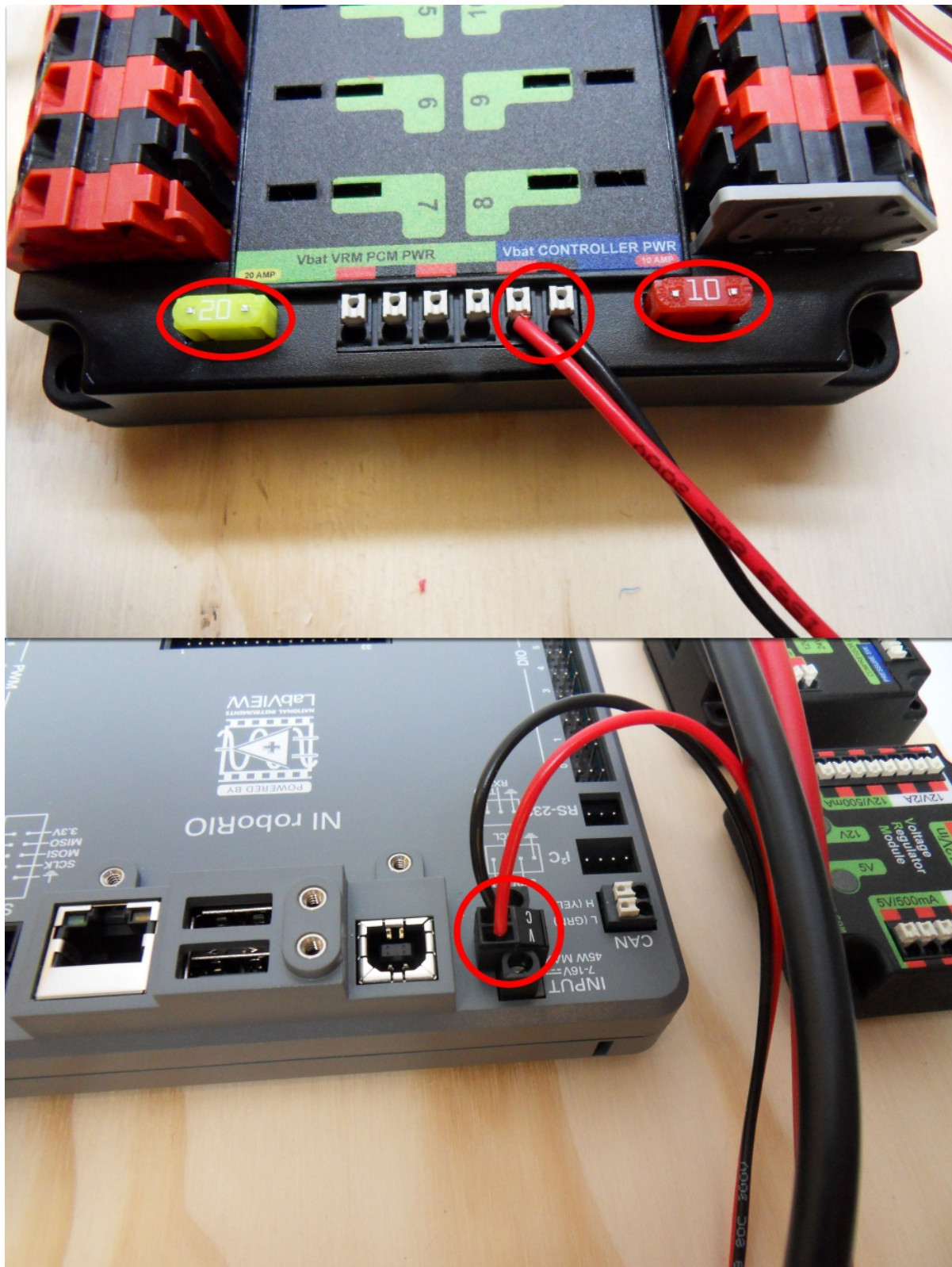
- O fio deve estar entre 16AWG e 24AWG (consulte as regras e verifique cada espessura de fio correspondente ao componente que será utilizado)

- Deve-se desencapar aproximadamente 5/16" (~8mm) no final dos fios.
- Para inserir ou remover os fios, pressione o botão correspondente para abrir o terminal.

Depois de conectar, cheque para ter certeza se está tudo certo:

- Verifique se não à “pequenos fios” expostos para fora do terminal.
- Puxe o fio para verificar se ele está bem preso. Se ele ceder, abra o terminal e prenda-o melhor ou desencape um pouco mais a ponta do fio.

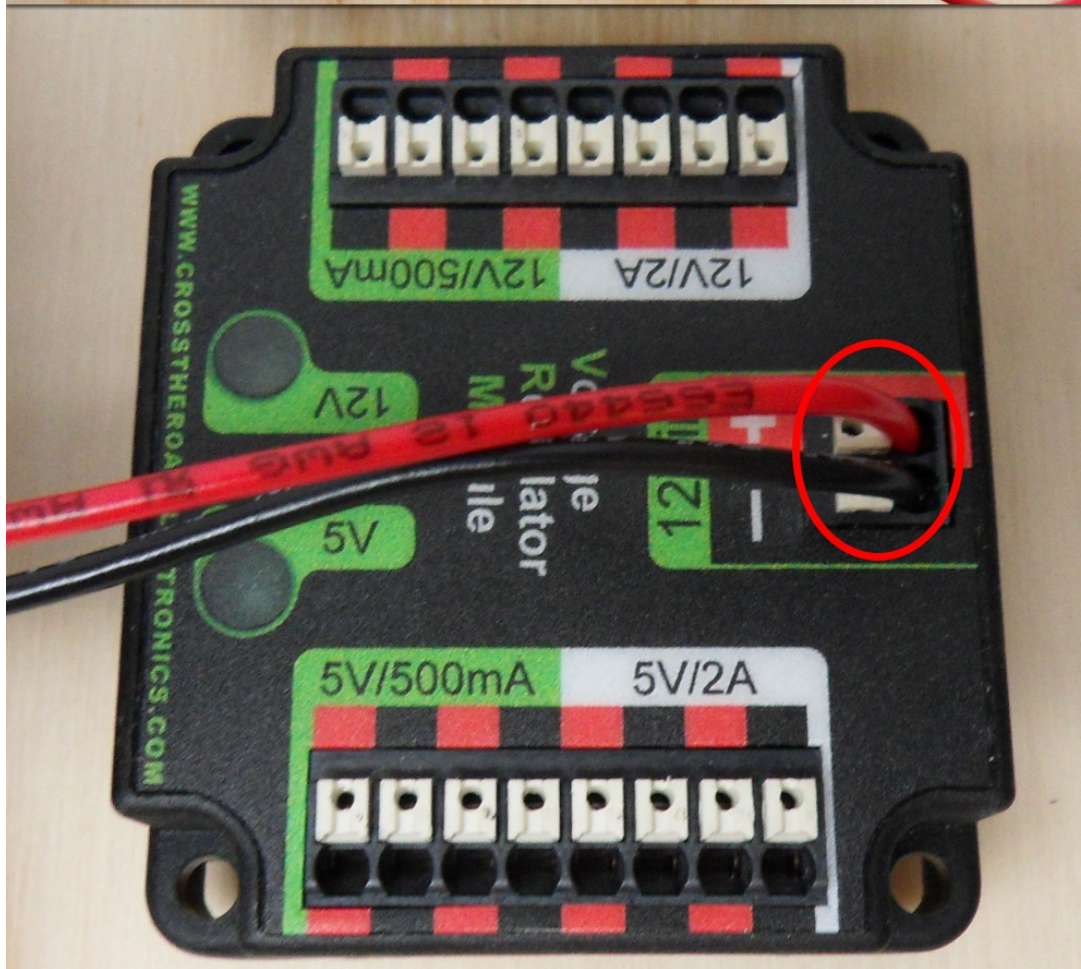
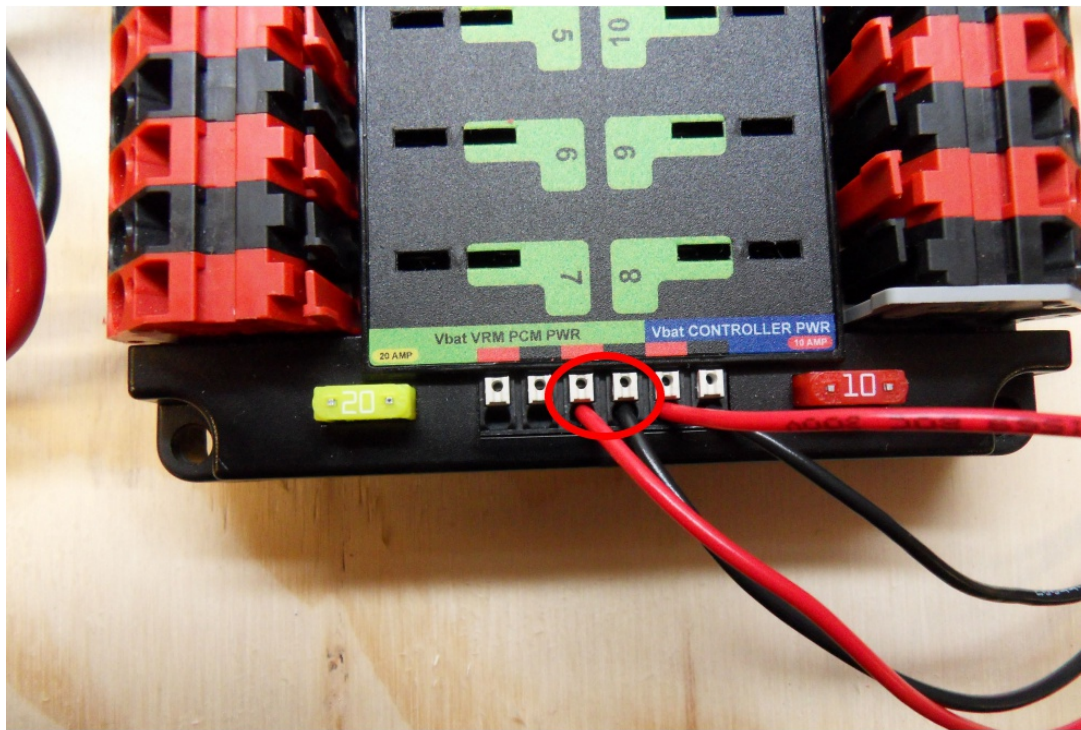
1.7.11 Energizando RoboRIO



****Requer:** Fusíveis automotivos de 10A e 20A, alicate desemcapador, chave fenda bem pequena, fios vermelho e preto de 18 AWG (1mm de seção) ******

1. Insira os fusíveis de 10 e 20 amperes na PDP nos lugares demonstrados na imagem acima.
2. Desencape as pontas dos fios de 18 AWG (1mm de seção) vermelho e preto e conecte aos terminais “Vbat Controller PWR” na PDP.
3. Meça o comprimento necessário dos fios para chegarem do RoboRio a PDP. Tome cuidado ao fazer o caminho desses fios, não passe por nenhum lugar que possa causar problemas.
4. Corte e desencape-os, adicionando o conector tubular em suas pontas.
5. Utilizando uma pequena chave de fenda, conecte os fios no terminal do RoboRIO (Vermelho no V e preto no C). Tenha certeza que o terminal está bem preso ao RoboRIO.

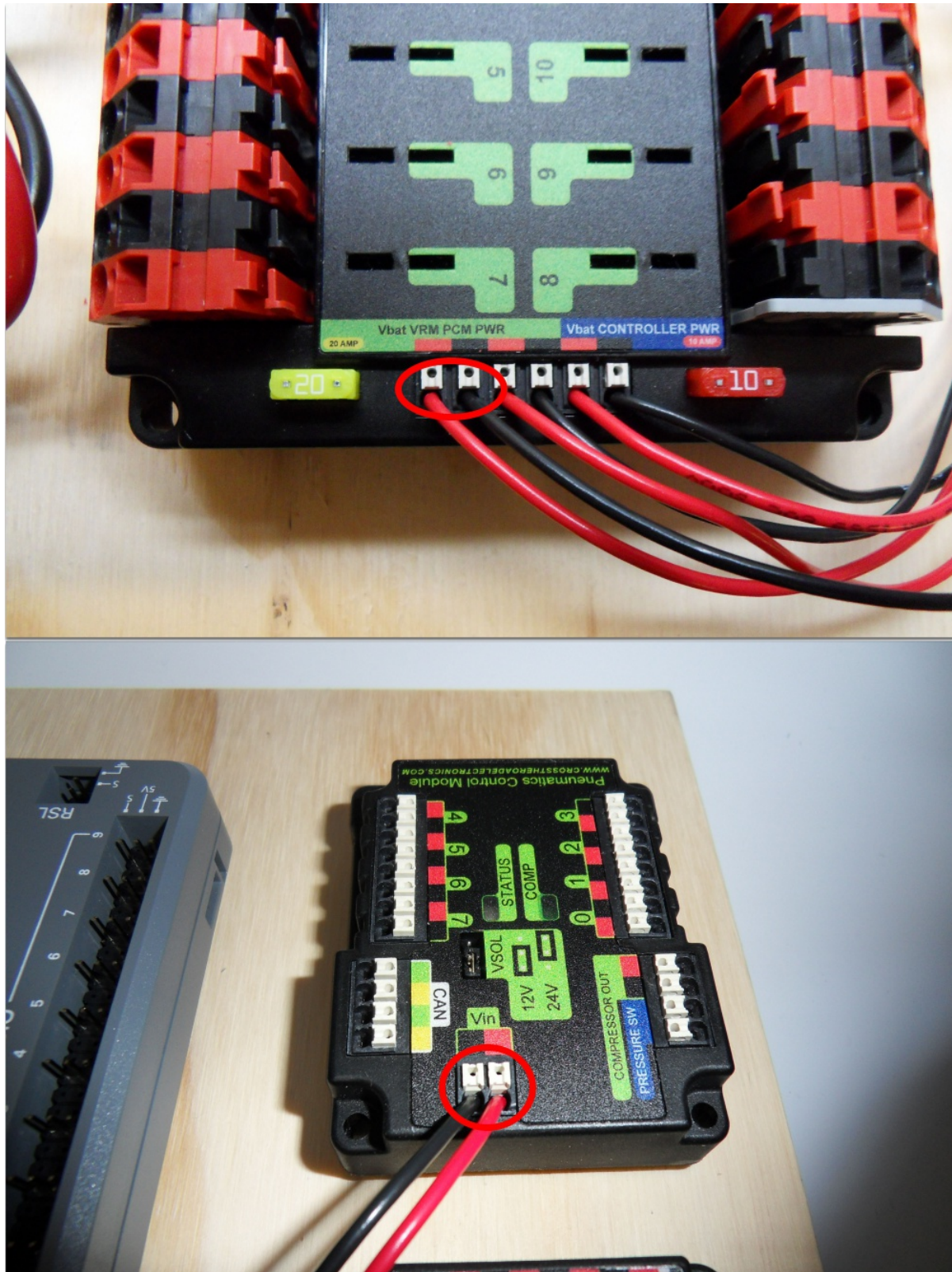
1.7.12 Energizando Voltage Regulator Module (VRM)



Requer: alicate desemcapador, chave de fenda pequena, fio vermelho e preto de 18 AWG (1mm de seção) red and black wire:

1. Desencape ~5/16" da ponta de um cabo vermelho e um cabo preto 18 AWG (1mm de seção).
2. Conecte os fios à um dos dois pares de terminais da PDP com o nome de "Vbat VRM PCM PWR".
3. Meça o tamanho necessário para chegar ao terminal "Vin" da PCM. Tome cuidado ao fazer o caminho desses fios, não passe por nenhum lugar que possa causar problemas.
4. Corte os cabos do tamanho correto e desencape as pontas encapadas.
5. Conecte os fios ao terminal 12Vin da VRM.

1.7.13 Energizando Pneumatics Control Module (PCM) (opcional)



Requer: Alicates, alicate de fenda pequena, chave de fenda pequena, cabo vermelho e preto de 18 AWG (1mm de seção)

Note: A PCM é um componente opcional, utilizado para controlar os sistemas pneumáticos do robô.

1. Desencape $\sim 5/16''$ ($\sim 8\text{mm}$) da ponta de um cabo vermelho e um cabo preto de 18 AWG (1mm de seção).
2. Conecte os fios à um dos dois pares de terminais da PDP com o nome de "Vbat VRM PCM PWR".
3. Meça o tamanho necessário para chegar ao terminal "Vin" da PCM. Tome cuidado ao fazer o caminho desses fios, não passe por nenhum lugar que possa causar problemas.
4. Corte os cabos do tamanho correto e desencape as pontas encapadas.
5. Conecte os fios ao terminal 12Vin da PCM.

1.7.14 Ethernet e energia do rádio

Warning: NÃO conecte o POE diretamente ao RoboRIO. Você deve utilizar um CABO ETHERNET que faça a ponte entre o POE e o RoboRIO.



Requer: Chave de fenda pequena, Cabo PoE incluso no Kit

1. Insira os conectores do PoE nos terminais correspondentes a 12V/2A da VRM.

2. Conecte o conector macho de Ethernet (RJ45) à porta de ethernet do rádio mais próxima da entrada do pino de energia (com o nome de 18-24v POE).

ATENÇÃO: Não conecte mais nada no barramento 12V/2A da VRM, as 2 portas devem ser utilizadas apenas e exclusivamente para o rádio.

1.7.15 Conectando o rádio ao RoboRIO

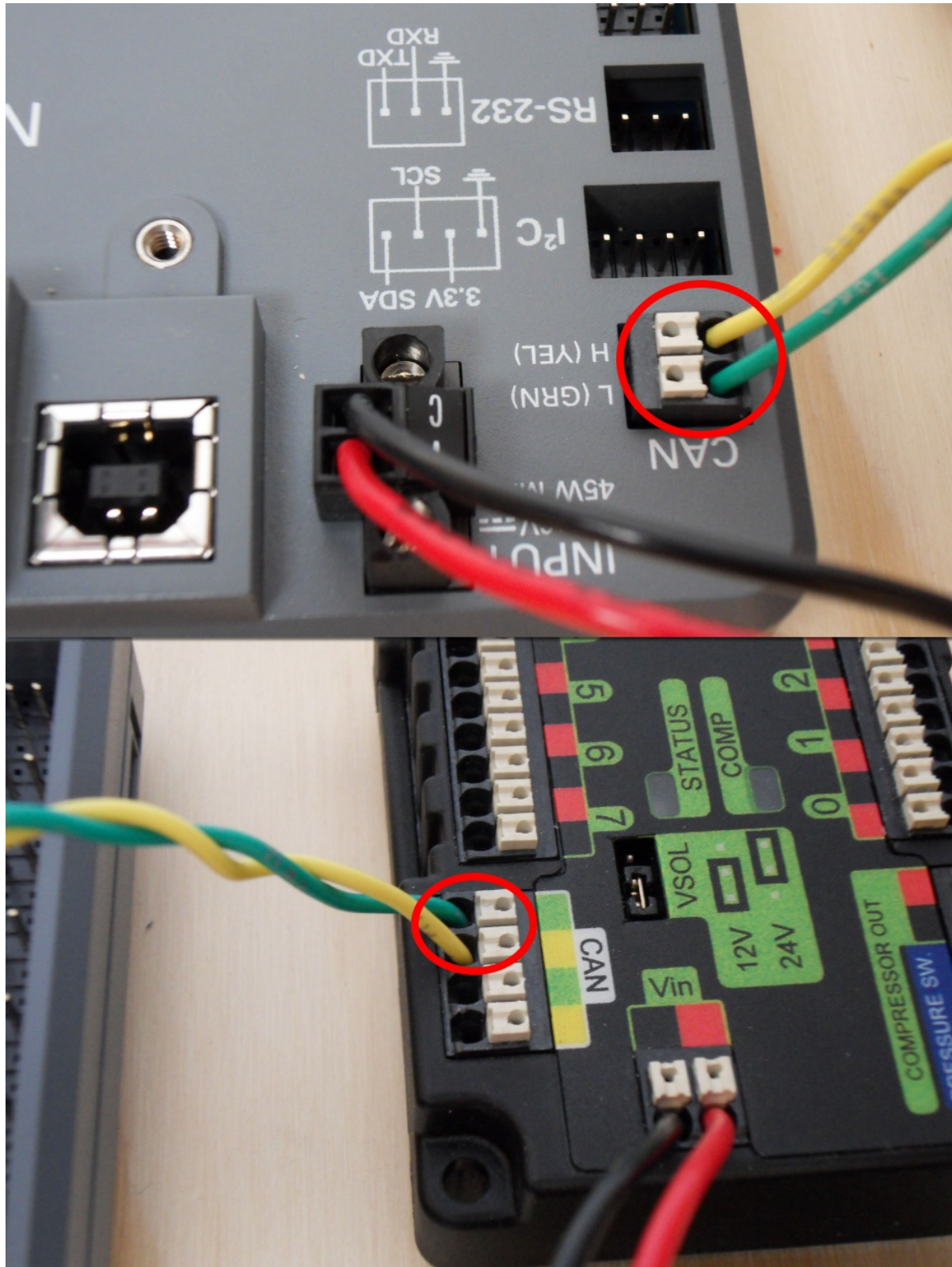


Requer: Cabo Ethernet

Conecte o cabo de Ethernet na porta RJ45 do cabo POE e na entrada Ethernet (RJ45) do RoboRIO.

1.7.16 Dispositivos CAN

CAN do RoboRIO para a PCM



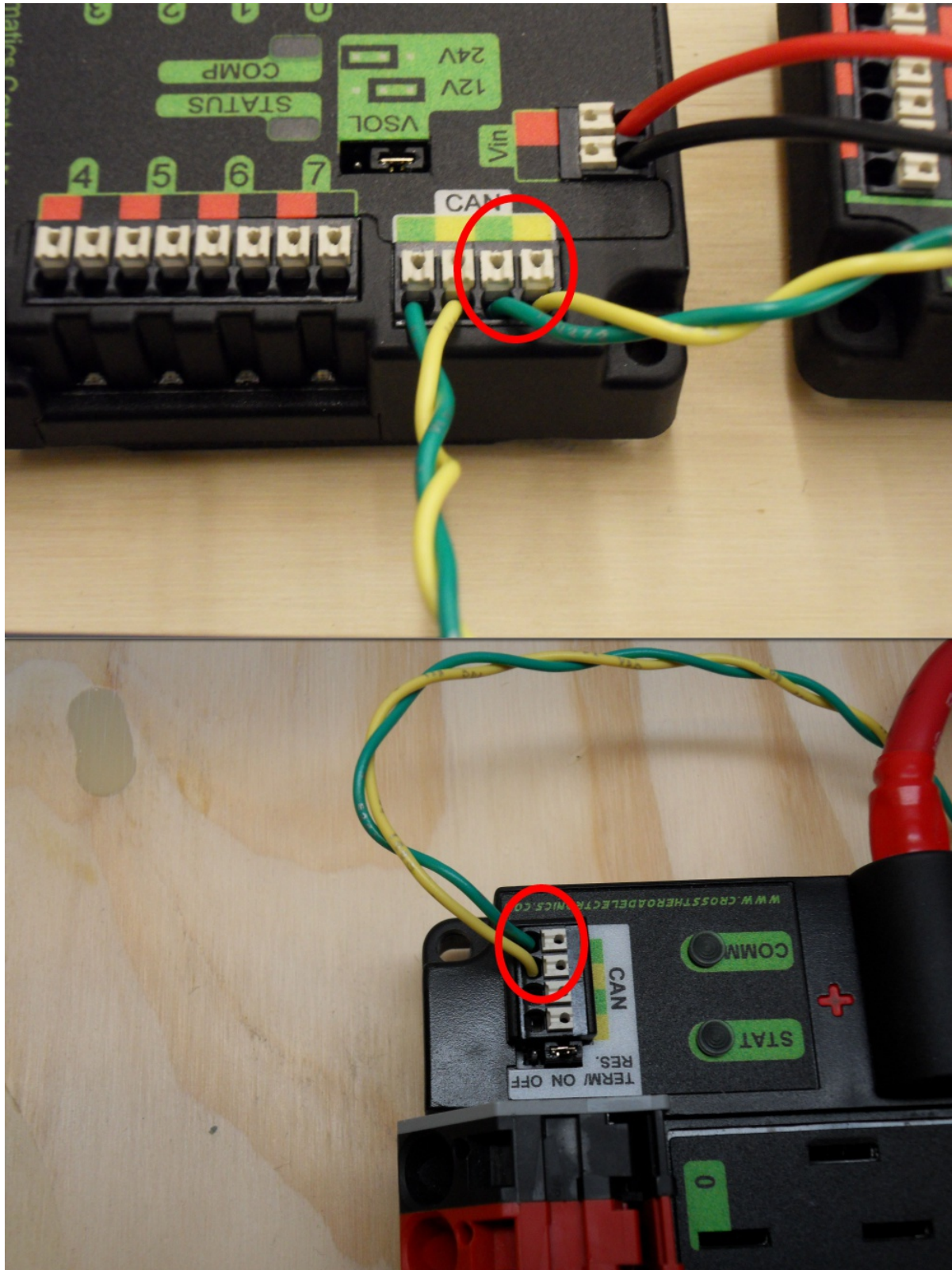
Requer : Alicates desmcapador, chave de fenda pequena, fios verde e amarelo de 22

AWG

A PCM é um componente opcional, utilizado para controlar os sistemas pneumáticos do robô. Se não estiver utilizando uma PCM, conecte os fios CAN que vem do RoboRIO diretamente na PDP.

1. Desencape ~5/16”(8mm) de cada fio CAN.
2. Insira os fios nos terminais destinados à linha CAN no RoboRIO (Amarelo -> YEL, Verde -> GRN).
3. Meça o comprimento necessário dos cabos para chegar com um pouco de folga na PCM.
4. Insira os fios CAN no terminal destinado à eles. Pode utilizar qualquer um dos pares de terminais CAN (Amarelo/Verde).

CAN da PCM para a PDP



Requer : Alicate desemcapador, chave de fenda pequena, fios verde e amarelo de 22

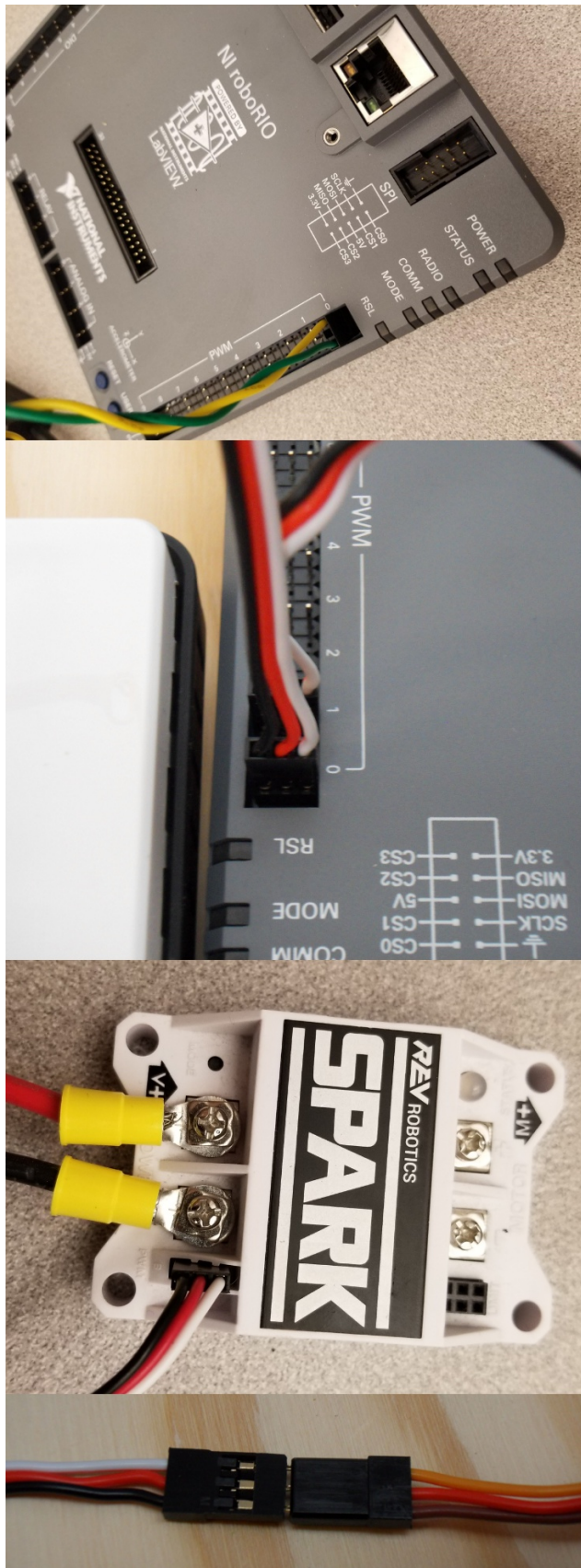
AWG

A PCM é um componente opcional, utilizado para controlar os sistemas pneumáticos do robô. Se não estiver utilizando uma PCM, conecte os fios CAN que vem do RoboRIO diretamente na PDP.

1. Insira os fios nos terminais CAN da PCM
2. Meça o comprimento dos fios para que consigam chegar até a PDP (cada um deles).
3. Insira os fios nos terminais CAN da PDP. Pode usar qualquer um dos dois pares da entrada.

Utilize a PDP como o último componente da linha CAN (CAN bus)

1.7.17 Cabos PWM



Requer: 4x PWM se não estiver utilizando controladores com cabos integrados, 2x PWM Y-cable (Opcional)

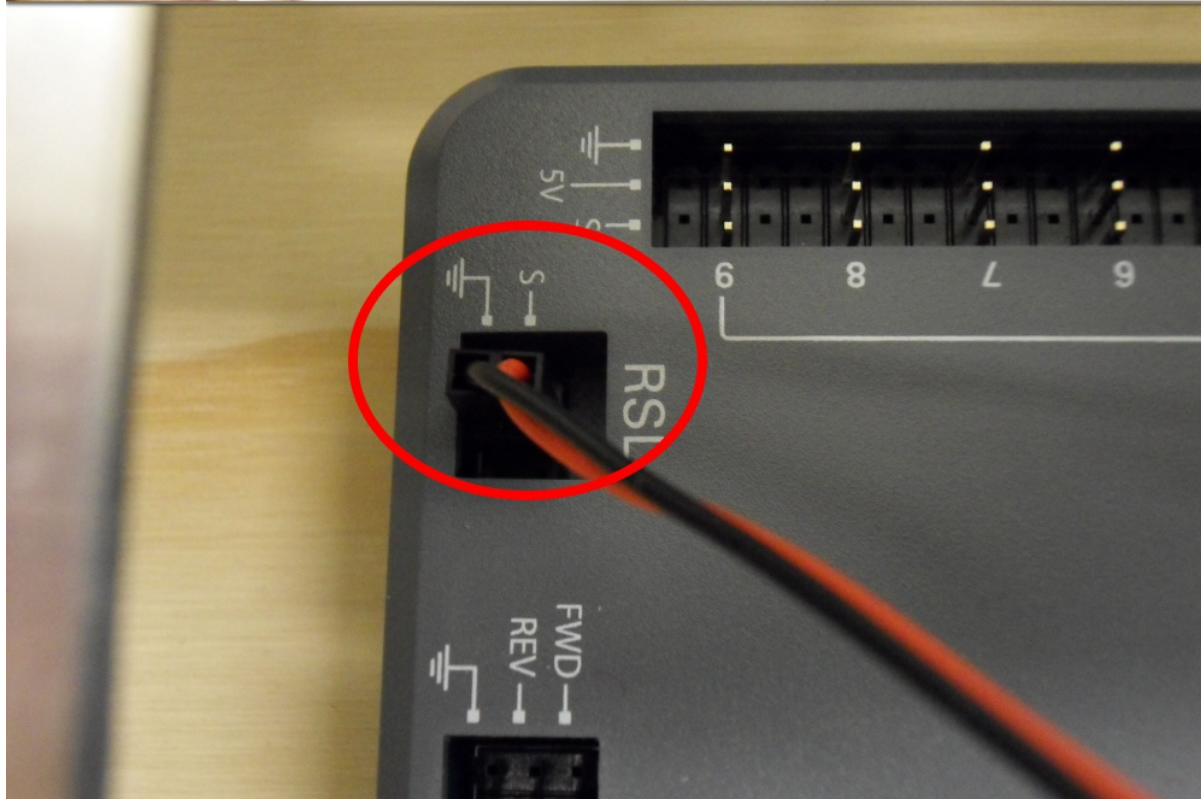
Opção 1 (conectar diretamente):

- Conecte os cabos PWM de cada controlador diretamente no RoboRIO. Para o Victor SPX e outros controladores PWM/CAN, o fio verde (fio preto para controladores com fios não integrados) deve estar conectado na parte mais próxima da borda do RoboRIO. Para controladores com fios não integrados, tenha certeza de que o fio preto esteja conectado conforme o controlador indica.

Opção 2 (cabo Y)

1. Conecte 1 cabo PWM Y aos cabos PWM dos controladores que controlam um lado do robô. O fio marrom do cabo Y deve corresponder ao fio verde/preto do cabo PWM do controlador.
2. Conecte os cabos PWM Y às portas PWM do RoboRIO. O fio marrom deve ser conectado ao lado mais próximo à borda do RoboRIO. É recomendado que se conecte o lado esquerdo ao PWM 0 e o lado direito ao PWM 1 para ajudar na organização durante a programação, mas fique atento: As entradas devem estar de acordo com o lado do robô que lhes for designado.

1.7.18 Robot Signal Light (RSL)



Requer: Alicates desmcapador, cabo de 2 vias, Robot Signal Light, fio vermelho de

18 AWG (1mm de seção) e chave de fenda pequena.

1. Desencape a ponta de um cabo preto e um cabo vermelho e prenda um conector tubular em cada um;
2. Insira o cabo preto no terminal do centro (N), e prenda-o (aperte o parafuso do terminal);
3. Corte um pequeno cabo vermelho 18 AWG (1mm de seção), insira uma das pontas no terminal “La” e prenda-o, insira a outra ponta no terminal “Lb”, mas ainda não prenda;
4. Insira o cabo vermelho com o conector tubular no terminal “Lb”, junto com o cabo vermelho pequeno e prenda-os;
5. Conecte os cabos com conector tubular na porta RSL do RoboRIO. O fio preto deve ser conectado à porta mais próxima da borda do RoboRIO.

Você deve prender (temporariamente) a RSL à chapa/base utilizando zip ties ou Dual Shock (é muito importante que a RSL esteja presa em um local bem visível do robô).

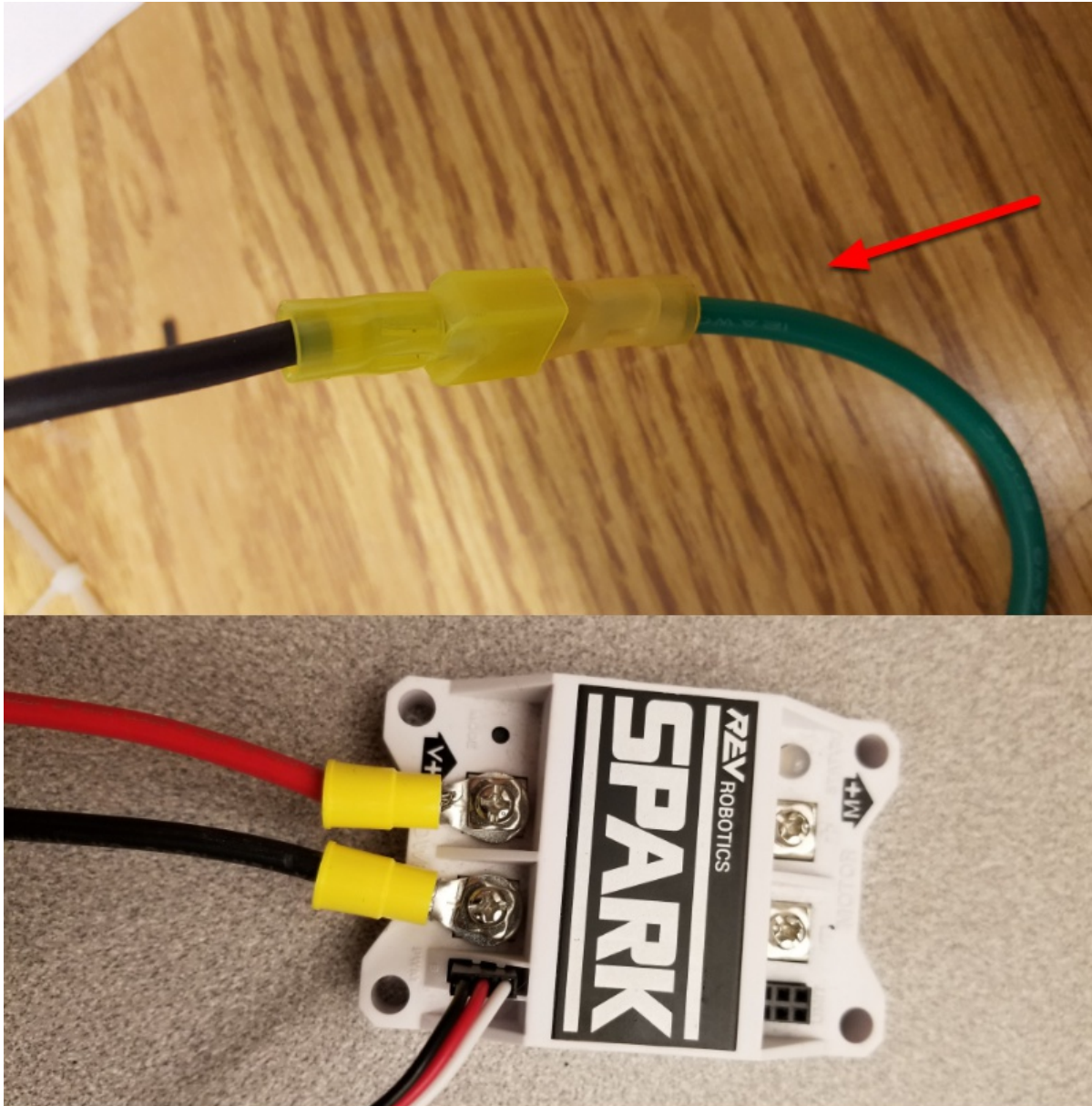
1.7.19 Circuit Breakers / Fusíveis



Requer: 4x 40A circuit breakers (fusível de 40 amperes)

Adicione um fusível de 40 amperes na posição correspondente onde os cabos do controlador estão conectados à PDP. Observe que as frestas para encaixar o fusível sempre estão ao lado da entrada positiva correspondente. Todos os terminais negativos estão conectados internamente..

1.7.20 Energizando motor



****Requer:** Alicates desmcapador, crimpador, chave phillips, terminais de conexão entre cabos (terminais olho por exemplo).

Para cada CIM motor:

- Estique os fios vermelhos e pretos de cada motor CIM

Para controladores com fios integrados (incluindo Victor SPX):

1. Deixe os cabos brancos e vermelhos do controlador esticados.
2. Conecte os fios do motor nos fios output/saída do controlador (é recomendado que conecte o fio vermelho ao fio branco M+).

Para Sparks e outros controladores com fios não integrados:

1. Crimpe um terminal olho ou garfo em cada um dos fios (tanto dos motores quanto controladores).
2. Conecte os fios no output/saída de cada controlador de motor (vermelho no positivo, preto no negativo).

1.7.21 STOP



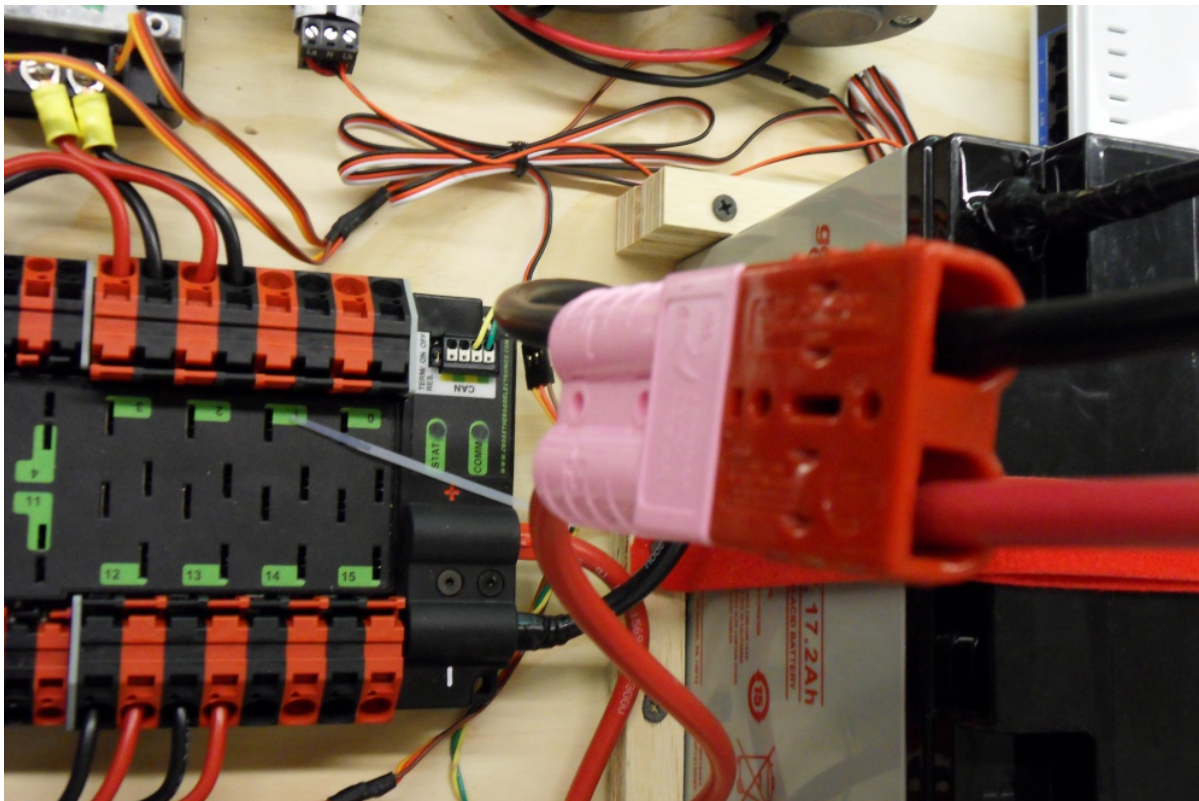
Danger: STOP!!

Danger: Antes de conectar a bateria, tenha certeza que todas as conexões estão com as polaridades corretas (positivo no positivo, negativo no negativo). Provavelmente devem ter algumas que não estão conectadas, cheque tudo.

- Verifique se o fio vermelho está conectado ao terminal positivo da bateria.
- Verifique se o fio vermelho está indo do main breaker para o terminal positivo da PDP e se o fio preto está indo para o terminal negativo.
- Para cada controlador de motor, verifique se o fio vermelho que sai do terminal vermelho da PDP está conectado com o fio vermelho do Victor SPX (não o M+ branco!!!!)
- Para cada componente que está ligado à PDP, verifique se o fio vermelho que sai do terminal vermelho da PDP se conecta com o terminal positivo do componente.
- Tenha certeza de que o conector PoE está conectado diretamente ao rádio, NÃO AO RoboRIO! Para conectar no RoboRIO, deve ser utilizado um cabo Ethernet adicional.

É recomendado que o robô esteja com as rodas suspensas para prevenir acidentes se o robô se movimentar de maneira inesperada.

1.7.22 Organize os fios



Requer: Zip ties

Essa é a hora de adicionar alguns zip ties aos fios. Isso fará com que os fios do robô se mantenham firmes e organizados.

Conecte a bateria ao robô com o conector Anderson. Para ligar o robô, empurre a “alavanca” do disjuntor principal até ela dar um estalo e ficar presa. Se as os eletrônicos começarem a piscar, provavelmente está tudo certo. Agora conecte o RoboRIO ao computador e tente passar o código!

1.8 Formatando seu roboRIO

Warning: Antes de criar imagens do seu roboRIO, você deve ter concluído a instalação do *Ferramentas de jogo FRC*. Você também deve ter a energia do roboRIO conectada corretamente ao painel de distribuição de energia. Verifique se os fios de energia do roboRIO estão seguros e se o conector está bem firme no roboRIO (4 parafusos no total para verificar).

1.8.1 Configurando o roboRIO

A ferramenta de imagem roboRIO será usada para criar uma imagem do seu roboRIO com o software mais recente.

Conexão USB



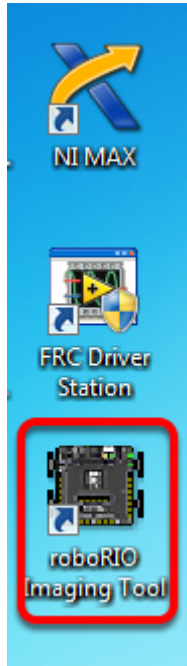
Conecte um cabo USB da porta do dispositivo USB roboRIO ao PC. Isso requer um cabo USB tipo A macho (extremidade padrão do PC) para tipo B cabo macho (quadrado com 2 cantos cortados), mais comumente encontrado como cabo USB da impressora.

Note: O roboRIO só deve ser visualizado através da conexão USB. Não é recomendável tentar criar imagens usando a conexão Ethernet.

Instalação do Driver

O Driver do dispositivo deve ser instalado automaticamente. Se você ver um pop-up “Dispositivo Novo” no canto inferior direito da tela, aguarde a instalação do driver antes de continuar.

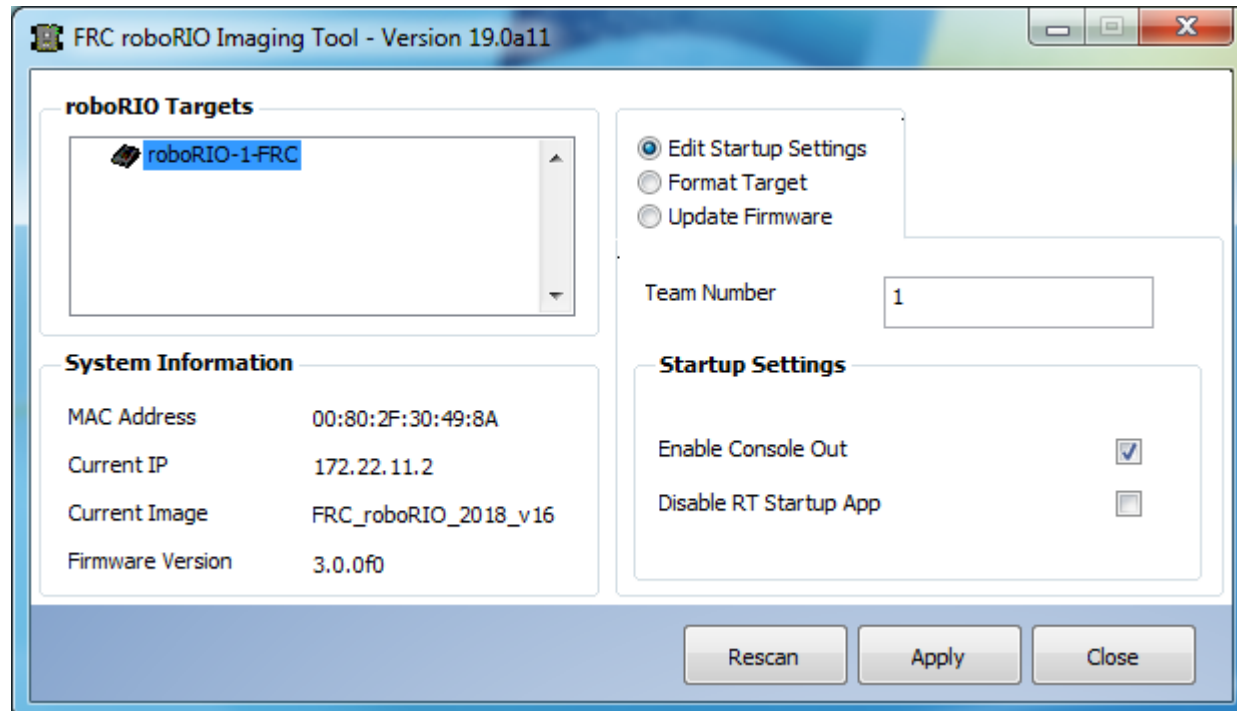
1.8.2 Iniciando a ferramenta de imagem



A ferramenta de imagem roboRIO e a imagem mais recente são instaladas com as ferramentas do jogo FRC da NI. Inicie a ferramenta de criação de imagens clicando duas vezes no atalho na área de trabalho. Se você tiver dificuldades em criar imagens do seu roboRIO, pode ser necessário clicar com o botão direito do mouse no ícone e selecionar Executar como administrador.

Note: A ferramenta de criação de imagens roboRIO também está localizada C:\Program Files (x86)\National Instruments\LabVIEW YYYY\project\roboRIO Tool\`onde YYYY é o ano atual - 1. Se for 2020, o diretório seria ``LabVIEW 2019.

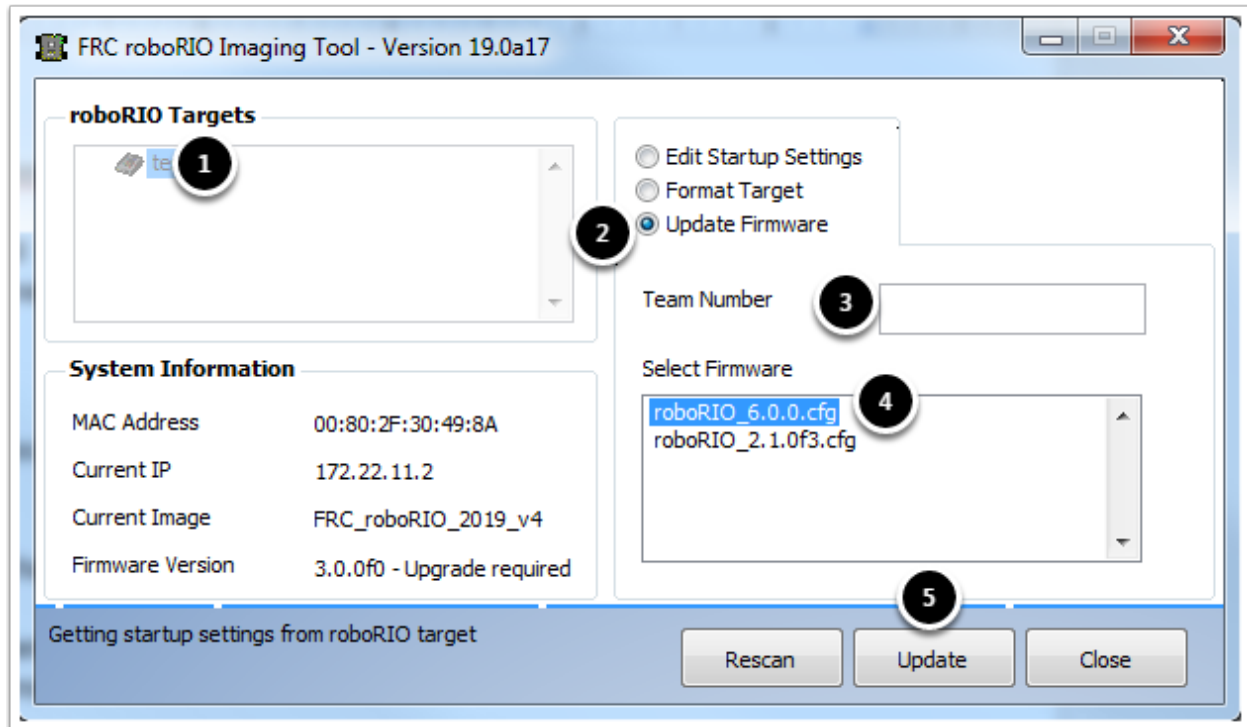
1.8.3 Ferramenta de imagem do roboRIO



Após do lançamento, a ferramenta de imagem roboRIO procurará os roboRIOs disponíveis e indicará qualquer um encontrado na caixa superior esquerda. A caixa inferior esquerda mostra informações e configurações do roboRIO selecionado atualmente. O painel direito contém controles para modificar as configurações do roboRIO:

- **Editar configurações de inicialização** - Esta opção é usada quando você deseja definir as configurações de inicialização do roboRIO (as configurações no painel direito), sem gerar imagens do roboRIO.
- **Formatar alvo** - Esta opção é usada quando você deseja carregar uma nova imagem no roboRIO (ou atualizar novamente a imagem existente). Essa é a opção mais comum.
- **Atualizar Firmware** - Esta opção é usada para atualizar o firmware do roboRIO. Para esta temporada, a ferramenta de criação de imagens exigirá que o firmware do roboRIO seja da versão 5.0 ou superior.

Atualização do Firmware

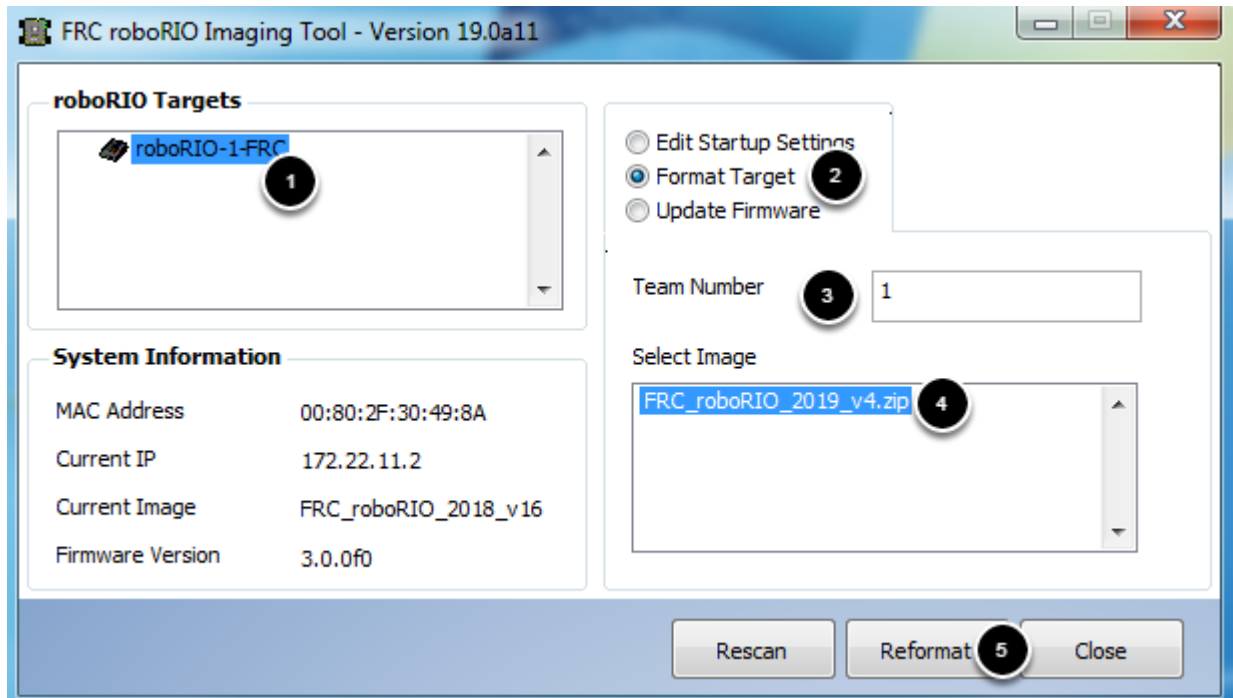


O firmware do roboRIO deve ter pelo menos a versão 5.0 para funcionar com a imagem de 2019. Se o seu roboRIO tiver pelo menos a versão 5.0 (como mostrado na parte inferior esquerda da ferramenta de criação de imagens), você não precisará atualizar.

Para atualizar o firmware do roboRIO:

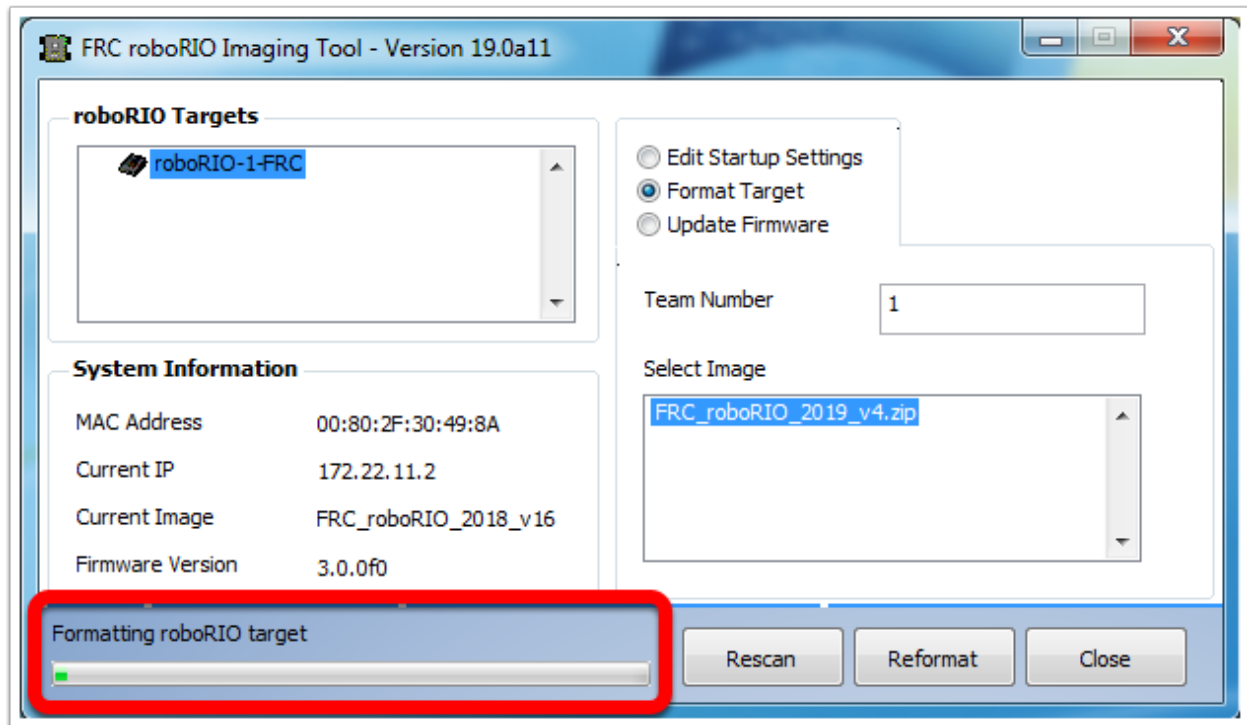
1. Verifique se o seu roboRIO está selecionado no painel superior esquerdo.
2. Selecione atualizar firmware no painel superior direito.
3. Digite um número de equipe na caixa Número da equipe.
4. Selecione o arquivo firmware mais recente no canto inferior direito.
5. Clique no botão **atualizar**.

1.8.4 Visualizando o roboRIO



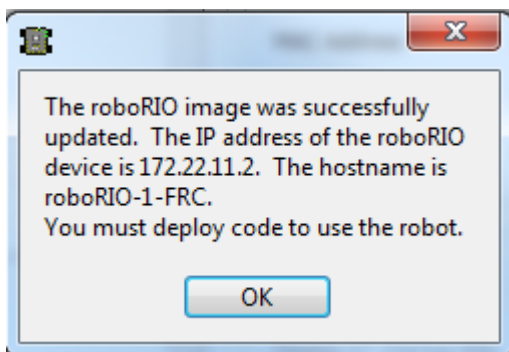
1. Verifique se o roboRIO está selecionado no painel superior esquerdo.
2. Selecione formatar alvo no painel direito.
3. Digite o número da sua equipe na caixa.
4. Selecione a versão mais recente da imagem na caixa.
5. Clique em reformatar para iniciar o processo de visualização da imagem.

1.8.5 Progresso de visualização da imagem



O processo de visualização da imagem levará aproximadamente de 3 a 10 minutos. Uma barra de progresso no canto inferior esquerdo da janela indicará o progresso.

1.8.6 Visualização completa



Quando a imagem for concluída, você deverá ver a caixa de diálogo acima. Clique em Ok e clique no botão Fechar no canto inferior direito para fechar a ferramenta de criação de imagens. Reinicie o roboRIO usando o botão Redefinir para que o novo número da equipe entre em vigor.

Note: A funcionalidade padrão de webdash CAN foi removida da imagem (os dispositivos CAN ainda funcionarão com o código do robô). Você precisará usar as ferramentas fornecidas por fornecedores individuais para atender seus dispositivos CAN.

1.8.7 Solução de problemas

Se você não conseguir criar uma imagem do seu roboRIO, as etapas de solução de problemas incluem:

- Tente executar a ferramenta de imagem roboRIO como administrador clicando com o botão direito do mouse no ícone da área de trabalho para iniciá-la.
- Tente acessar a página do roboRIO com um navegador da web em `http://172.22.11.2/` e / ou verifique se o adaptador de rede da NI aparece na sua lista de adaptadores de rede no painel de controle. Caso contrário, tente reinstalar o NI FRC Game Tools ou tente um PC diferente.
- Verifique se o seu firewall está desligado.
- Experimente um PC diferente.
- Algumas equipes enfrentam um problema em que a criação de imagens falha se o nome do dispositivo do computador que você está usando tiver um traço (""). Tente renomear o computador (ou usando um PC diferente).
- Tente inicializar o roboRIO no modo de segurança pressionando e segurando o botão de reset por pelo menos 5 segundos.

1.9 Programming your Radio

This guide will show you how to use the FRC Radio Configuration Utility software to configure your robot's wireless bridge for use outside of FRC events.

Before you begin using the software:

1. Disable WiFi connections on your computer, as it may prevent the configuration utility from properly communicating with the bridge
2. Make sure no devices are connected to your computer via ethernet, other than the wireless bridge.

Warning: The OM5P-AN and AC use the same power plug as the D-Link DAP1522, however they are 12V radios. Wire the radio to the 12V 2A terminals on the VRM (center-pin positive).

1.9.1 Pre-Requisites

The FRC Radio Configuration Utility requires the Java Runtime Engine (JRE). If you do not have Java installed, you can download the JRE from here: <https://www.java.com/en/download/>

The FRC Radio Configuration Utility requires Administrator privileges to configure the network settings on your machine. The program should request the necessary privileges automatically (may require a password if run from a non-Administrator account), but if you are having trouble, try running it from an Administrator account.

1.9.2 Application Notes

By default, the Radio Configuration Utility will program the radio to enforce the 4Mbps bandwidth limit on traffic exiting the radio over the wireless interface. In the home configuration (AP mode) this is a total, not a per client limit. This means that streaming video to multiple clients is not recommended.

The Utility has been tested on Windows 7, 8 and 10. It may work on other operating systems, but has not been tested.

Programmed Configuration

Power	
Blue	On or Powering Up
Blue Blinking	Powering Up
Eth Link	
Blue	Link Up
Blue Blinking	Traffic Present
WiFi	
	Bridge Mode, Unlinked or non-FRC firmware
Off	
Red	AP, Unlinked
Yellow\Orange	AP, Linked
Green	Bridge Mode, Linked

WiFi light only works after radio has been power cycled.



The Radio Configuration Utility programs a number of configuration settings into the radio when run. These settings apply to the radio in all modes (including at events). These include:

- Set a static IP of 10.TE.AM.1
- Set an alternate IP on the wired side of 192.168.1.1 for future programming
- Bridge the wired ports so they may be used interchangeably
- The LED configuration noted in the graphic above.
- 4Mb/s bandwidth limit on the outbound side of the wireless interface (may be disabled for home use)
- QoS rules for internal packet prioritization (affects internal buffer and which packets to discard if bandwidth limit is reached). These rules are:
 - Robot Control and Status (UDP 1110, 1115, 1150)
 - Robot TCP & Network Tables (TCP 1735, 1740)
 - Bulk (All other traffic). (disabled if BW limit is disabled)
- DHCP server enabled. Serves out:
 - 10.TE.AM.11 - 10.TE.AM.111 on the wired side

- 10.TE.AM.130 - 10.TE.AM.230 on the wireless side
- Subnet mask of 255.255.255.0
- Broadcast address 10.TE.AM.255
- DNS server enabled. DNS server IP and domain suffix (.lan) are served as part of the DHCP.

At home only:

- SSID may have a “Robot Name” appended to the team number to distinguish multiple networks.
- Firewall option may be enabled to mimic the field firewall rules (open ports may be found in the Game Manual)

Warning: It is not possible to modify the configuration manually

Download the software Download the latest FRC Radio Configuration Utility Installer from the following links:

[FRC Radio Configuration 20.0.0](#)

[FRC Radio Configuration 20.0.0 Israel Version](#)

Note: The _IL version is for Israel teams and contains a version of the OM5PAC firmware with restricted channels for use in Israel.

Warning: Version 19.1.1 corrects an issue with applying the Bandwidth Limit present in version 19.1.0. Teams should install the new version, then re-program their radio (there is no need to re-flash the firmware).

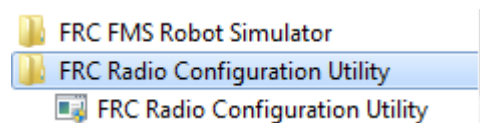
1.9.3 Install the software

File Name	Date/Time	Type
FRC_Radio_Configuration_10_8_15.exe	10/8/2015 1:50 PM	Application
FRCicon_RGB_Border.bmp	2/20/2015 1:27 PM	Bitmap Image

Double click on FRC_Radio_Configuration_VERSION.exe to launch the installer. Follow the prompts to complete the installation.

Part of the installation prompts will include installing Npcap if it is not already present. The Npcap installer contains a number of checkboxes to configure the install. You should leave the options as the defaults.

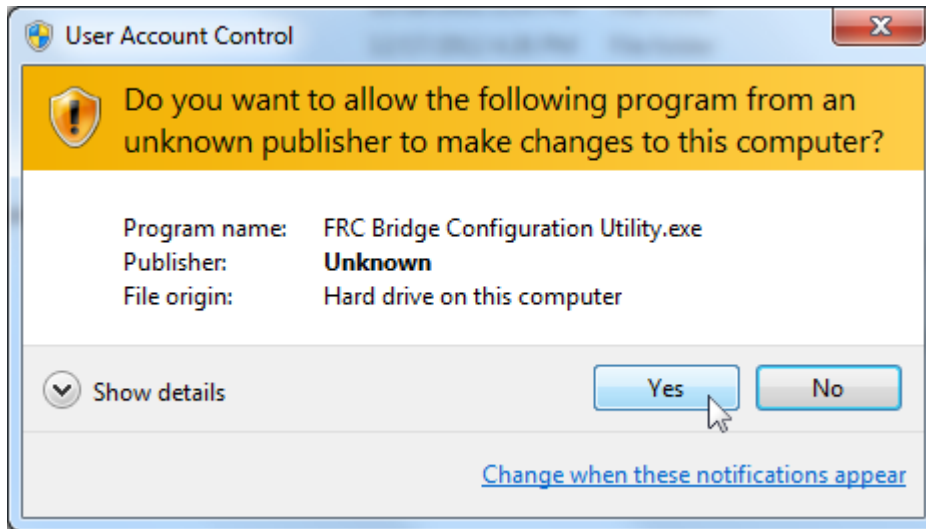
1.9.4 Launch the software



Use the Start menu or desktop shortcut to launch the program.

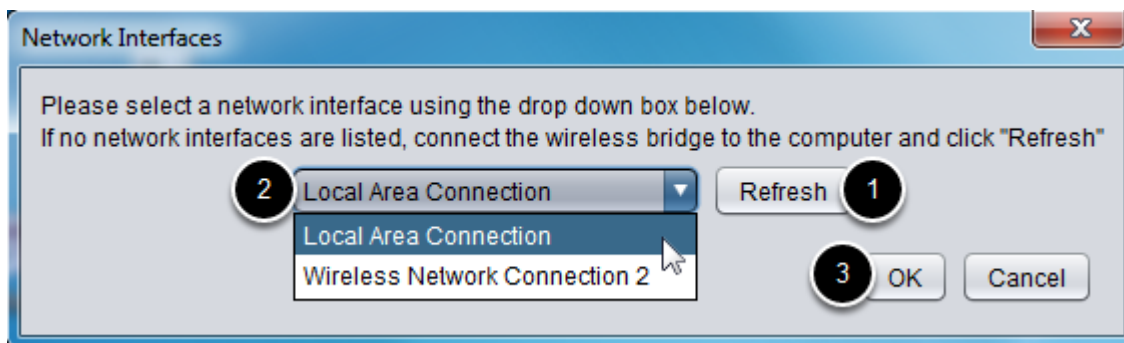
Note: If you need to locate the program it is installed to C:\Program Files (x86)\FRC Radio Configuration Utility. For 32-bit machines the path is C:\Program Files\FRC Radio Configuration Utility

1.9.5 Allow the program to make changes, if prompted



If the your computer is running Windows Vista or Windows 7, a prompt may appear about allowing the configuration utility to make changes to the computer. Click "Yes" if the prompt appears.

1.9.6 Select the network interface



Use the pop-up window to select the which ethernet interface the configuration utility will use to communicate with the wireless bridge. On Windows machines, ethernet interfaces are typically named "Local Area Connection". The configuration utility can not program a bridge over a wireless connection.

1. If no ethernet interfaces are listed, click "Refresh" to re-scan for available interfaces
2. Select the interface you want to use from the drop-down list
3. Click "OK"

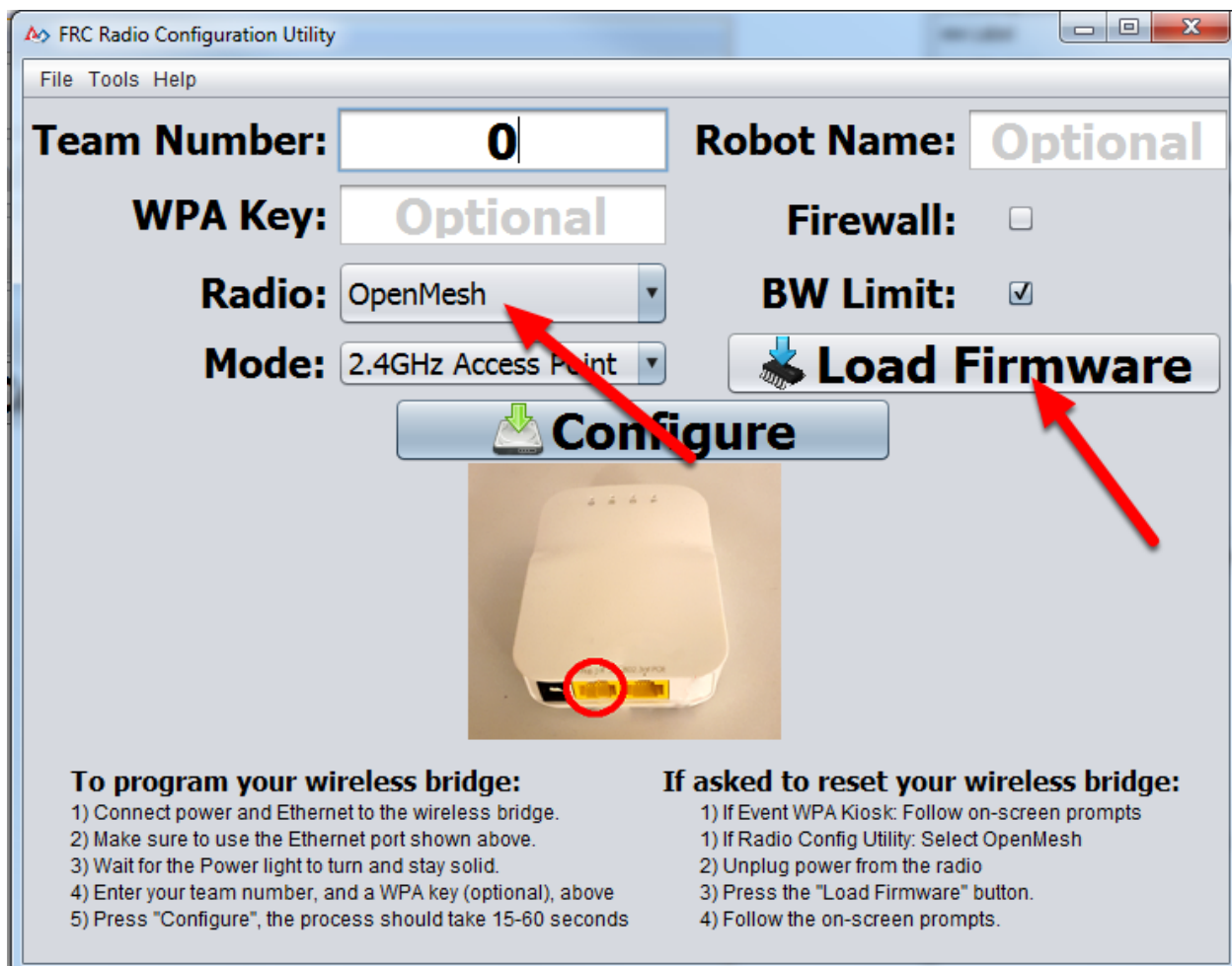
1.9.7 Open Mesh Firmware Note

For the FRC Radio Configuration Utility to program the OM5P-AN and OM5P-AC radio, the radio must be running an FRC specific build of the OpenWRT firmware. OM5P-AC radios in the 2019 KoP should not need an update.

If you do not need to update or re-load the firmware, skip the next step.

Warning: Note: Radios used in 2019 **do not** need to be updated before configuring, the 2020 tool uses the same 2019 firmware.

1.9.8 Loading FRC Firmware to OpenMesh radio



If you need to load the FRC firmware (or reset the radio), you can do so using the FRC Radio Configuration Utility.

1. Follow the instructions above to install the software, launch the program and select the Ethernet interface.
2. Make sure the OpenMesh radio is selected in the Radio dropdown.
3. Make sure the radio is connected to the PC via Ethernet.

4. Unplug the power from the radio. (If using a PoE cable, this will also be unplugging the Ethernet to the PC, this is fine)
5. Press the Load Firmware button
6. When prompted, plug in the radio power. The software should detect the radio, load the firmware and prompt you when complete.

Warning: If you see an error about NPF name, try disabling all adapters other than the one being used to program the radio. If only one adapter is found, the tool should attempt to use that one. See the steps in *“Troubleshooting: Disabling Network Adapters”* for more info.

Teams may also see this error with foreign language Operating Systems. If you experience issues loading firmware or programming on a foreign language OS, try using an English OS, such as on the KOP provided PC or setting the Locale setting to “en_us” as described on [this page](#).

1.9.9 Select a bridge model and operating mode

Team Number: **Robot Name:**

WPA Key: **Firewall:** ☐

Radio: **BW Limit:** ☒

Mode: **Load Firmware**

Configure

To program your wireless bridge:

- 1) Connect power and Ethernet to the wireless bridge.
- 2) Make sure to use the Ethernet port shown above.
- 3) Wait for the Power light to turn and stay solid.
- 4) Enter your team number, and a WPA key (optional), above
- 5) Press "Configure", the process should take 15-60 seconds

If asked to reset your wireless bridge:

- 1) If Event WPA Kiosk: Follow on-screen prompts
- 1) If Radio Config Utility: Select OpenMesh
- 2) Unplug power from the radio
- 3) Press the "Load Firmware" button.
- 4) Follow the on-screen prompts.

1. Select which radio you are configuring using the drop-down list.

2. Select which operating mode you want to configure. For most cases, the default selection of 2.4GHz Access Point will be sufficient. If your computers support it, the 5GHz AP mode is recommended, as 5GHz is less congested in many environments.

1.9.10 Select Options

FRC Radio Configuration Utility

File Tools Help

Team Number:

Robot Name:

WPA Key:

Radio:

Mode:

Firewall: ☐

BW Limit: ☒

Load Firmware

Configure

To program your wireless bridge:

- 1) Connect power and Ethernet to the wireless bridge.
- 2) Make sure to use the Ethernet port shown above.
- 3) Wait for the Power light to turn and stay solid.
- 4) Enter your team number, and a WPA key (optional), above
- 5) Press "Configure", the process should take 15-60 seconds

If asked to reset your wireless bridge:

- 1) If Event WPA Kiosk: Follow on-screen prompts
- 1) If Radio Config Utility: Select OpenMesh
- 2) Unplug power from the radio
- 3) Press the "Load Firmware" button.
- 4) Follow the on-screen prompts.

The default values of the options have been selected to match the use case of most teams, however, you may wish to customize these options to your specific scenario:

1. Robot Name: This is a string that gets appended to the SSID used by the radio. This allows you to have multiple networks with the same team number and still be able to distinguish them.
2. Firewall: If this box is checked, the radio firewall will be configured to attempt to mimic the port blocking behavior of the firewall present on the FRC field. For a list of open ports, please see the FRC Game Manual.
3. BW Limit: If this box is checked, the radio enforces a 4MB/s bandwidth limit like it does when programmed at events. Note that in AP mode, this is a total limit, not per client, so streaming video to multiple clients simultaneously may cause undesired behavior.

Note: Firewall and BW Limit only apply to the OpenMesh radios. These options have no

effect on D-Link radios.

Warning: The “Firewall” option configures the radio to emulate the field firewall. This means that you will not be able to deploy code wirelessly with this option enabled.

1.9.11 Prepare and start the configuration process

FRC Radio Configuration Utility

File Tools Help

Team Number: **Robot Name:**

WPA Key: **Firewall:** ☐

Radio: **BW Limit:** ☒

Mode: **Load Firmware**

Configure

To program your wireless bridge:

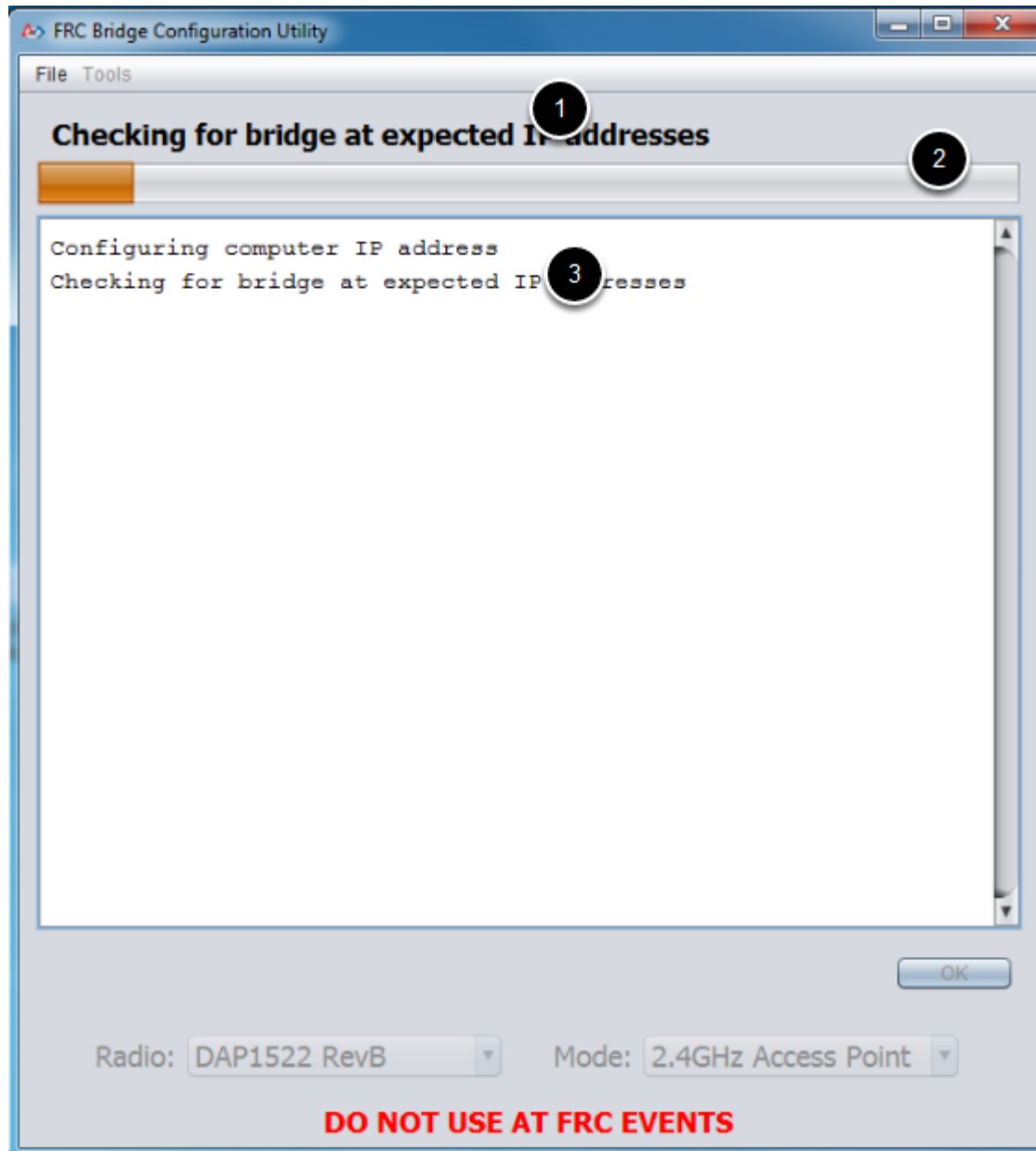
- 1) Connect power and Ethernet to the wireless bridge.
- 2) Make sure to use the Ethernet port shown above.
- 3) Wait for the Power light to turn and stay solid.
- 4) Enter your team number, and a WPA key (optional), above
- 5) Press "Configure", the process should take 15-60 seconds

If asked to reset your wireless bridge:

- 1) If Event WPA Kiosk: Follow on-screen prompts
- 1) If Radio Config Utility: Select OpenMesh
- 2) Unplug power from the radio
- 3) Press the "Load Firmware" button.
- 4) Follow the on-screen prompts.

Follow the on-screen instructions for preparing your wireless bridge, entering the settings the bridge will be configured with, and starting the configuration process. These on-screen instructions update to match the bridge model and operating mode chosen.

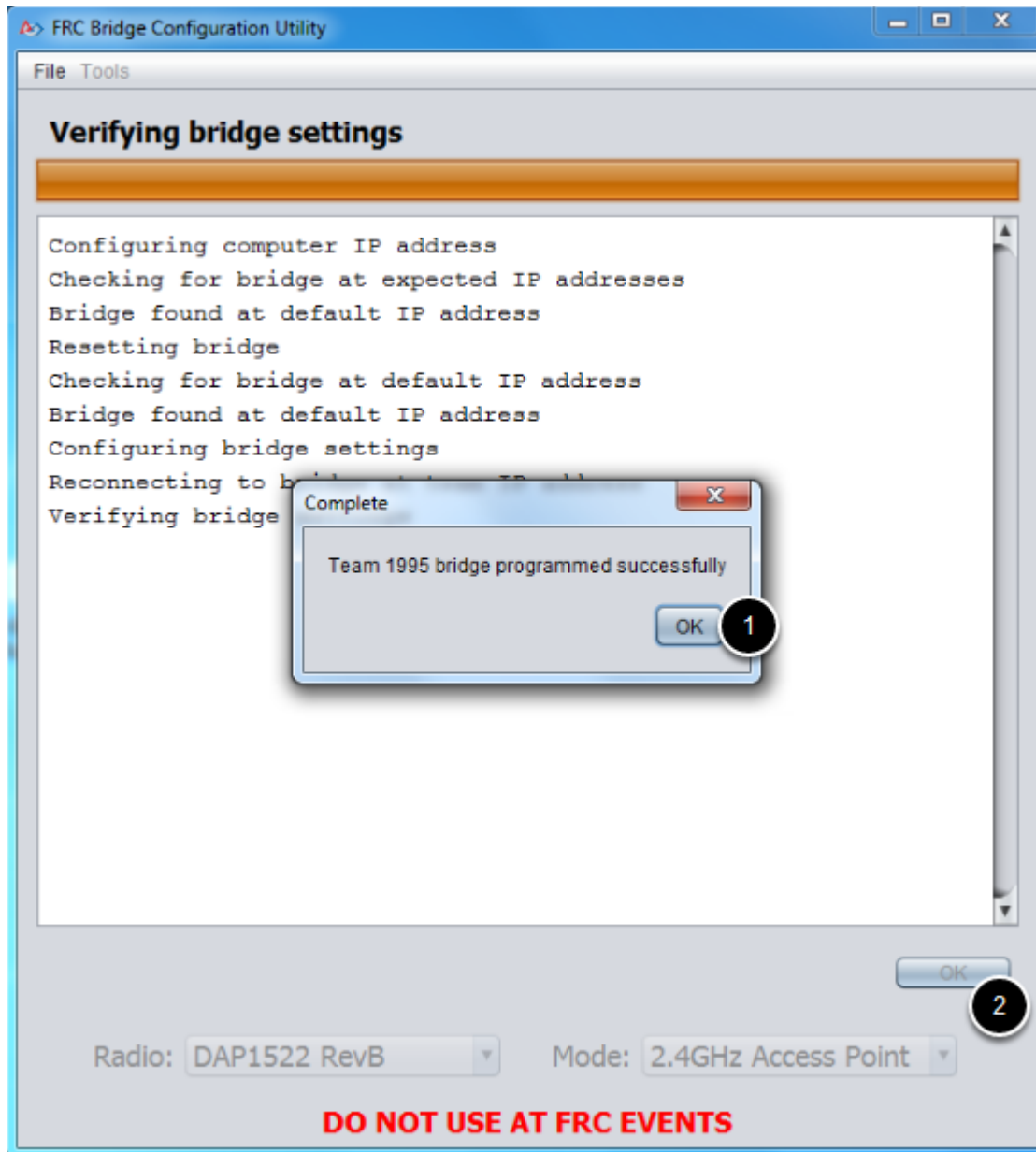
1.9.12 Configuration Progress



Throughout the configuration process, the window will indicate:

1. The step currently being executed
2. The overall progress of the configuration process
3. All steps executed so far

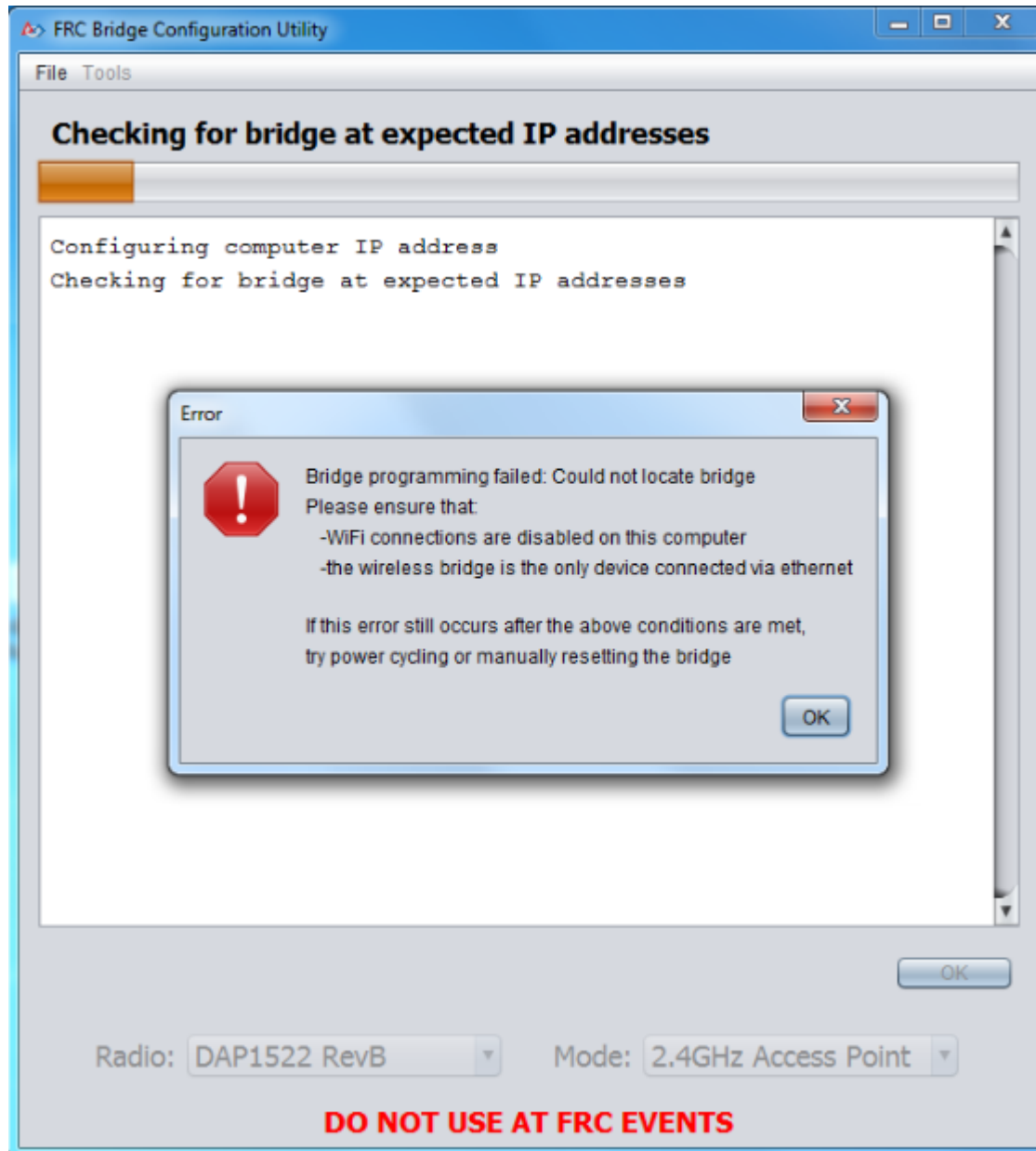
1.9.13 Configuration completed



Once the configuration is complete:

1. Press "OK" on the dialog window
2. Press "OK" on the main window to return to the settings screen

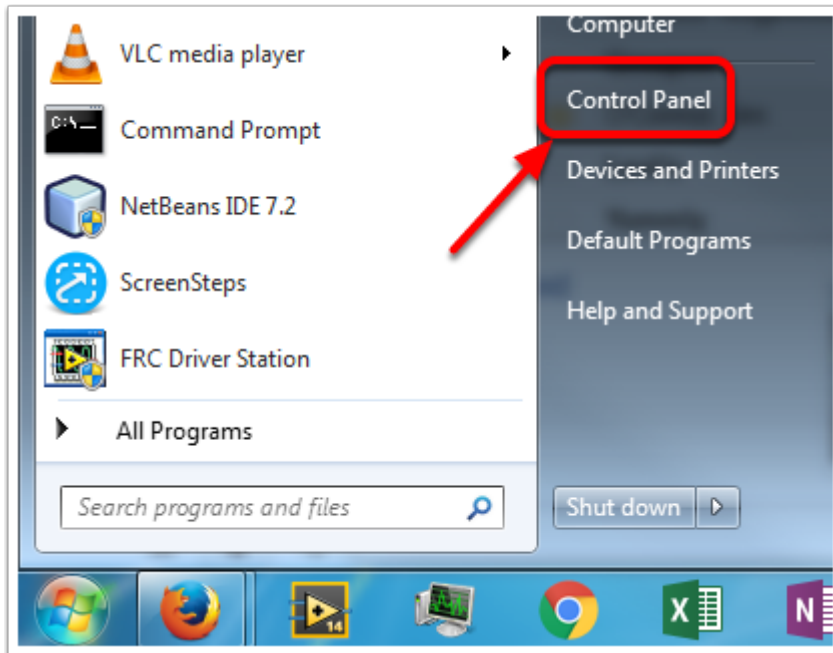
1.9.14 Configuration errors



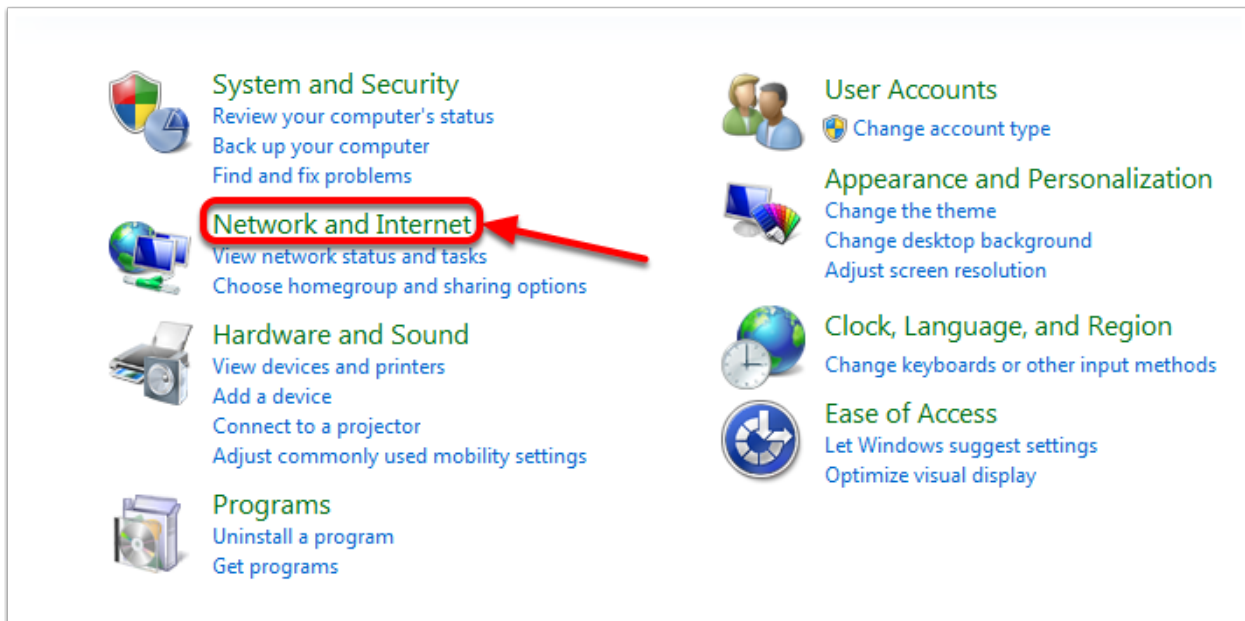
If an error occurs during the configuration process, follow the instructions in the error message to correct the problem.

1.9.15 Troubleshooting: Disabling Network Adapters

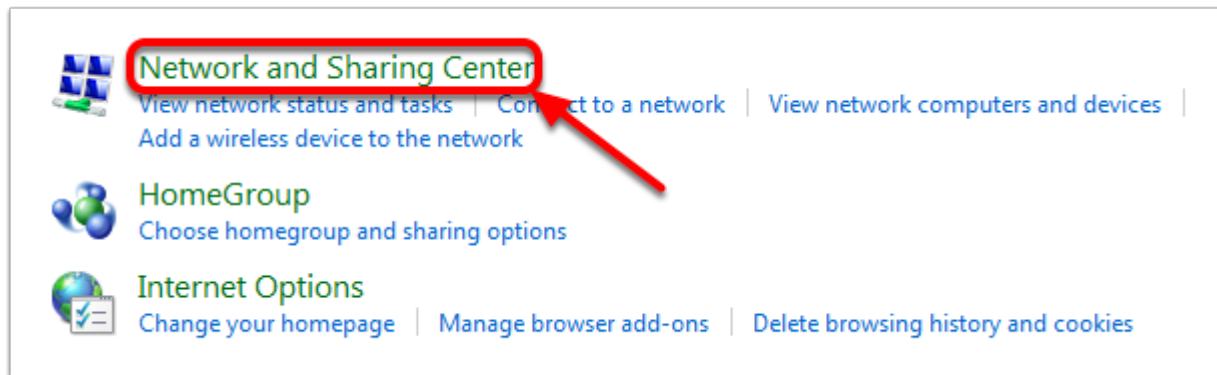
If you get an error message about "NPF adapter" when attempting to load firmware, you need to disable all other adapters. This is not always the same as turning the adapters off with a physical button or putting the PC into airplane mode. The following steps provide more detail on how to disable adapters.



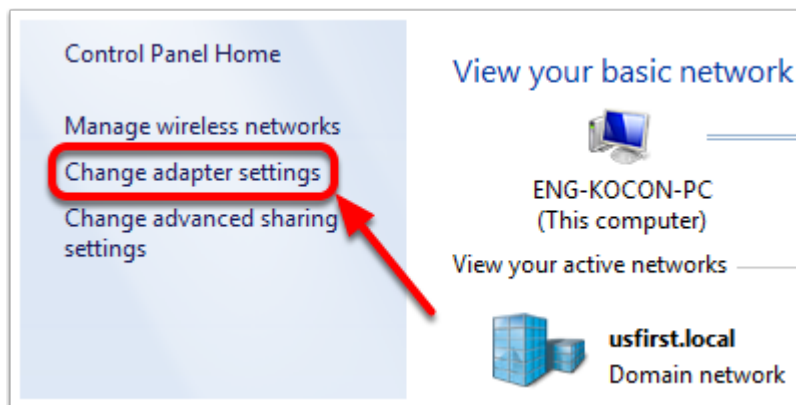
Open the Control Panel by going to Start->Control Panel



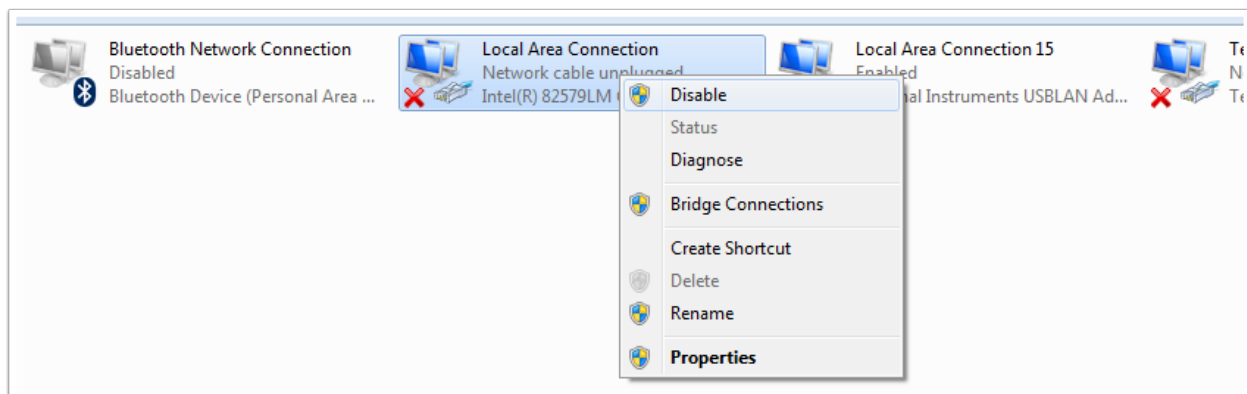
Choose the Network and Internet category.



Click Network and Sharing Center



On the left pane, click Change Adapter Settings



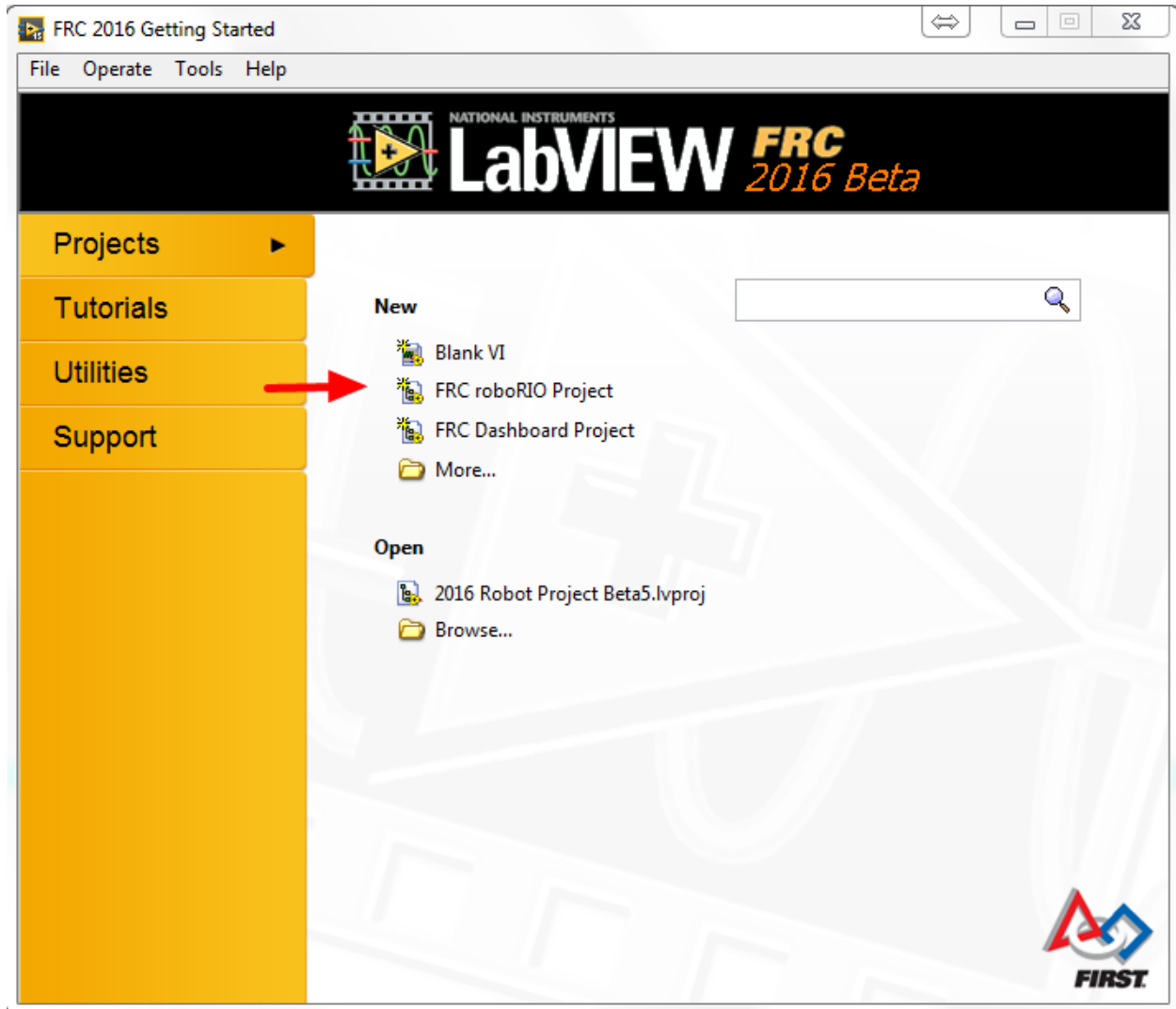
For each adapter other than the one connected to the radio, right click on the adapter and select Disable from the menu.

Getting Started with a Benchtop Robot

2.1 Creating your Benchtop Test Program (LabVIEW)

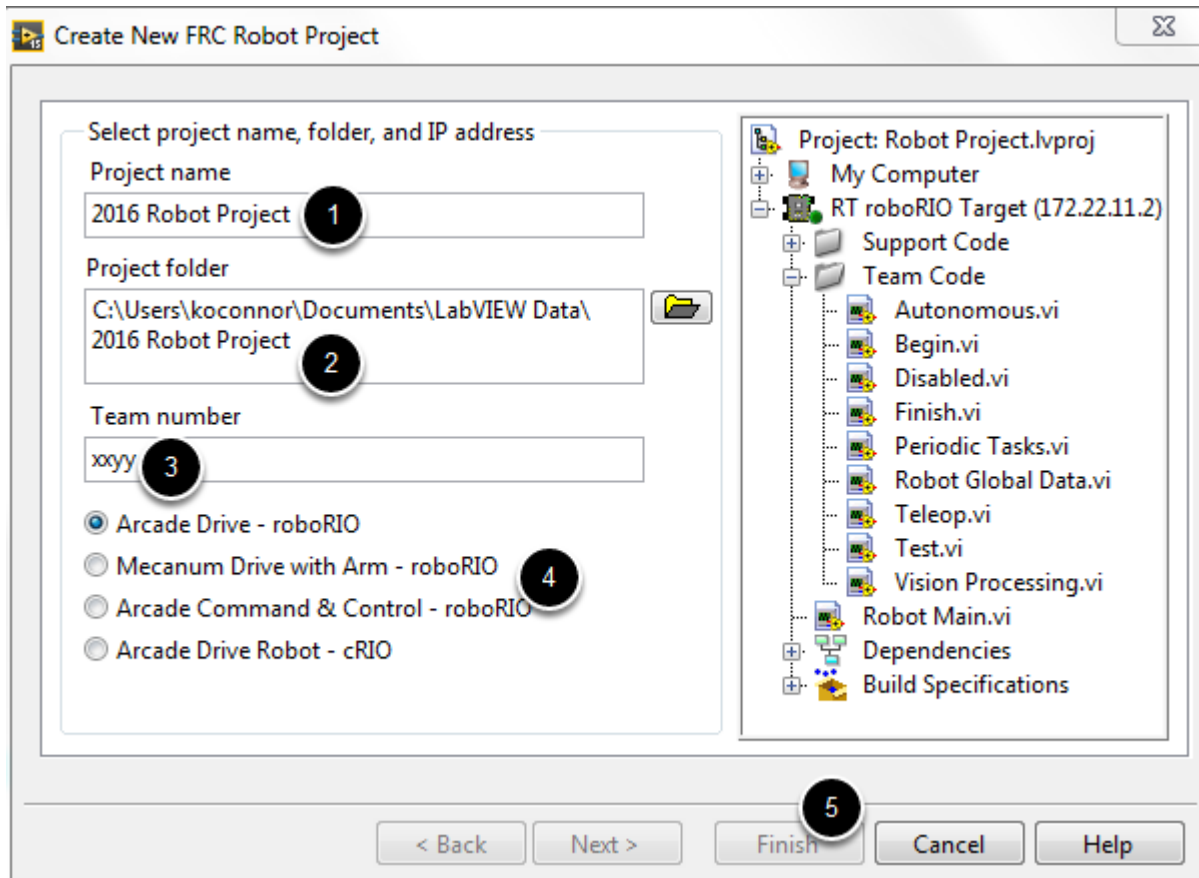
This document covers how to create, build and load an FRC LabVIEW program onto a roboRIO. Before beginning, make sure that you have installed LabVIEW for FRC and the FRC Driver Station and that you have configured and imaged your roboRIO as described previously.

2.1.1 Creating a Project



Launch LabVIEW and click the FRC roboRIO Robot Project link in the Projects window to display the Create New FRC Robot Project dialog box.

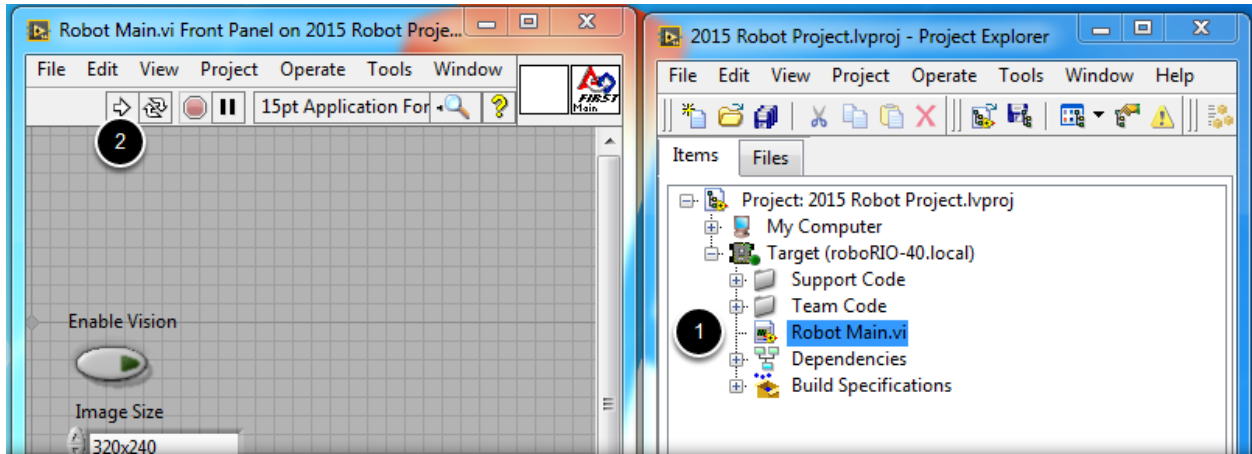
2.1.2 Configuring Project



Fill in the Create New FRC Project Dialog:

1. Pick a name for your project
2. Select a folder to place the project in.
3. Enter your team number
4. Select a project type. If unsure, select Arcade Drive - roboRIO.
5. Click Finish

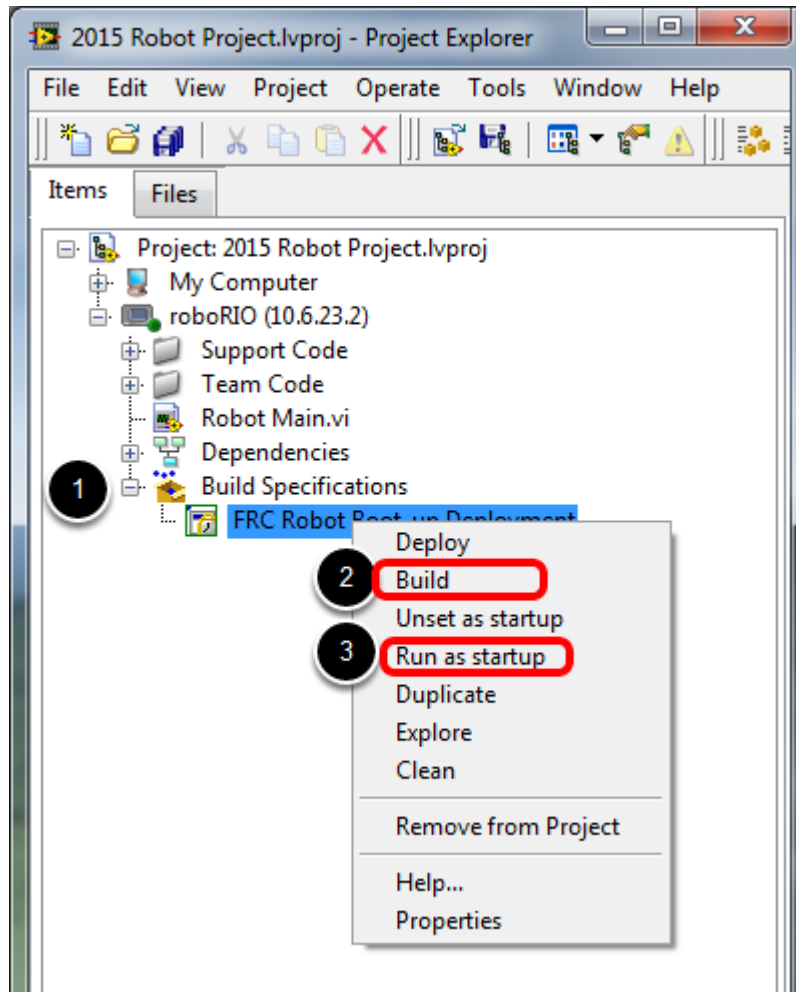
2.1.3 Running the Program



1. In the Project Explorer window, double-click the Robot Main.vi item to open the Robot Main VI.
2. Click the Run button (White Arrow on the top ribbon) of the Robot Main VI to deploy the VI to the roboRIO. LabVIEW deploys the VI, all items required by the VI, and the target settings to memory on the roboRIO. If prompted to save any VIs, click Save on all prompts.
3. Click the Abort button of the Robot Main VI. Notice that the VI stops. When you deploy a program with the Run button, the program runs on the roboRIO, but you can manipulate the front panel objects of the program from the host computer.

Note: A program deployed in this manner will not remain on the roboRIO after a power cycle. To deploy a program to run every time the roboRIO starts follow the next step, Deploying the program.

2.1.4 Deploying the program



To run in the competition, you will need to deploy a program to your roboRIO. This allows the program to survive across reboots of the controller, but doesn't allow the same debugging features (front panel, probes, highlight execution) as running from the front panel. To deploy your program:

1. In the Project Explorer, click the + next to Build Specifications to expand it.
2. Right-click on FRC Robot Boot-up Deployment and select Build. Wait for the build to complete.
3. Right-click again on FRC Robot Boot-Up Deployment and select Run as Startup. If you receive a conflict dialog, click OK. This dialog simply indicates that there is currently a program on the roboRIO which will be terminated/replaced.
4. Either check the box to close the deployment window on successful completion or click the close button when the deployment completes.
5. The roboRIO will automatically start running the deployed code within a few seconds of the dialog closing.

2.2 Running your Benchtop Test Program

2.2.1 Overview

You should create and download a Benchtop Test Program as described for your programming language:

C++/Java

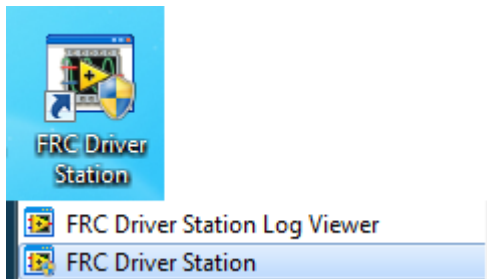
LabVIEW

2.2.2 Tethered Operation

Running your benchtop testing program while tethered to the Driver Station via ethernet or USB cable will confirm the the program was successfully deployed and that the driver station and roboRIO are properly configured.

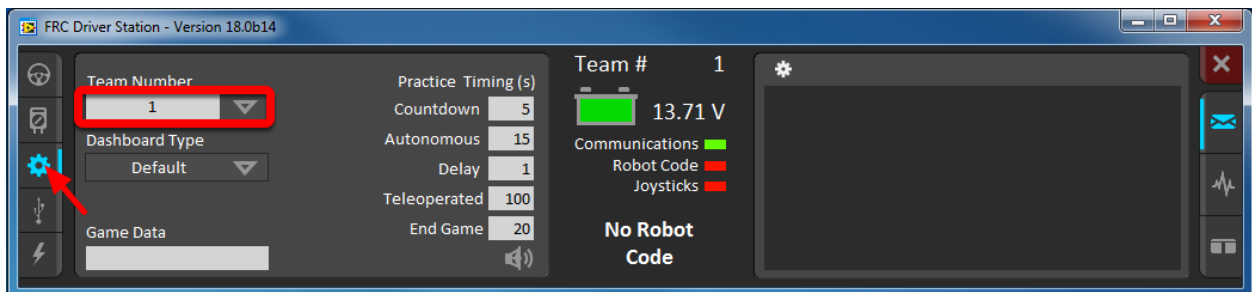
The roboRIO should be powered on and connected to the PC over Ethernet or USB.

2.2.3 Starting the FRC Driver Station



The FRC Driver Station can be launched by double-clicking the icon on the Desktop or by selecting Start->All Programs->FRC Driver Station.

2.2.4 Setting Up the Driver Station



The DS must be set to your team number in order to connect to your robot. In order to do this click the Setup tab then enter your team number in the team number box. Press return or click outside the box for the setting to take effect.

PCs will typically have the correct network settings for the DS to connect to the robot already, but if not, make sure your Network adapter is set to DHCP.

2.2.5 Confirm Connectivity

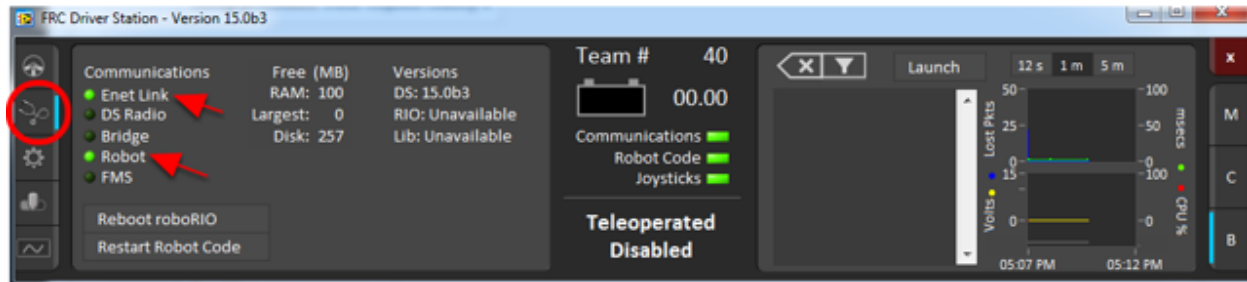


Fig. 1: Tethered

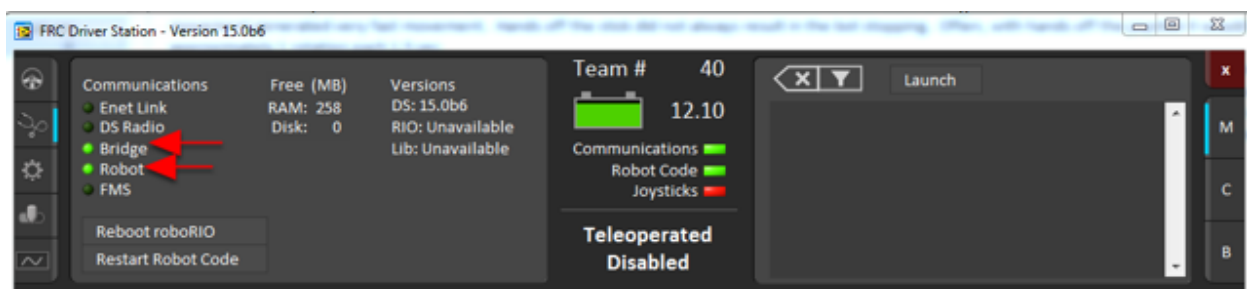
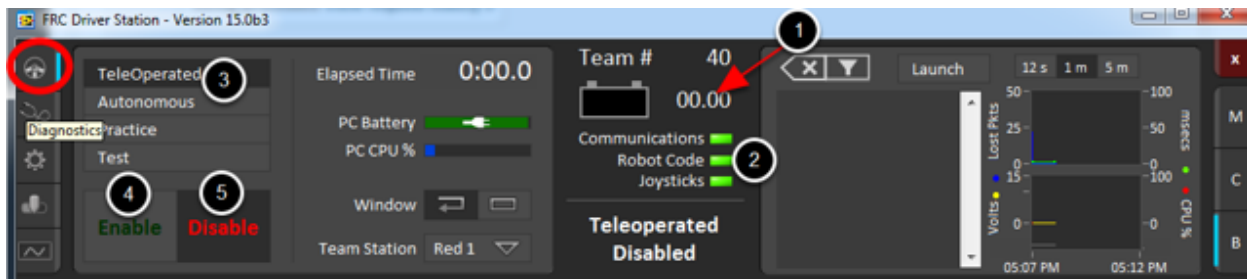


Fig. 2: Wireless

Using the Driver Station software, click Diagnostics and confirm that the Enet Link (or Robot Radio led, if operating wirelessly) and Robot leds are green.

2.2.6 Operate the Robot



Click the Operation Tab

1. Confirm that battery voltage is displayed
2. Communications, Robot Code, and Joysticks indicators are green.
3. Put the robot in Teleop Mode
4. Click Enable. Move the joysticks and observe how the robot responds.
5. Click Disable

2.2.7 Wireless Operation

Before attempting wireless operation, tethered operation should have been confirmed as described in *Tethered Operation*. Running your benchtop testing program while connected to the Driver Station via WiFi will confirm that the access point is properly configured.

Configuring the Access Point

See the article *Programming your radio* for details on configuring the robot radio for use as an access point.

After configuring the access point, connect the driver station wirelessly to the robot. The SSID will be your team number (as entered in the Bridge Configuration Utility). If you set a key when using the Bridge Configuration Utility you will need to enter it to connect to the network. Make sure the computer network adapter is set to DHCP (“Obtain an IP address automatically”).

You can now confirm wireless operation using the same steps in **Confirm Connectivity** and **Operate the Robot** above.

FRC LabVIEW Programming

3.1 Creating Robot Programs

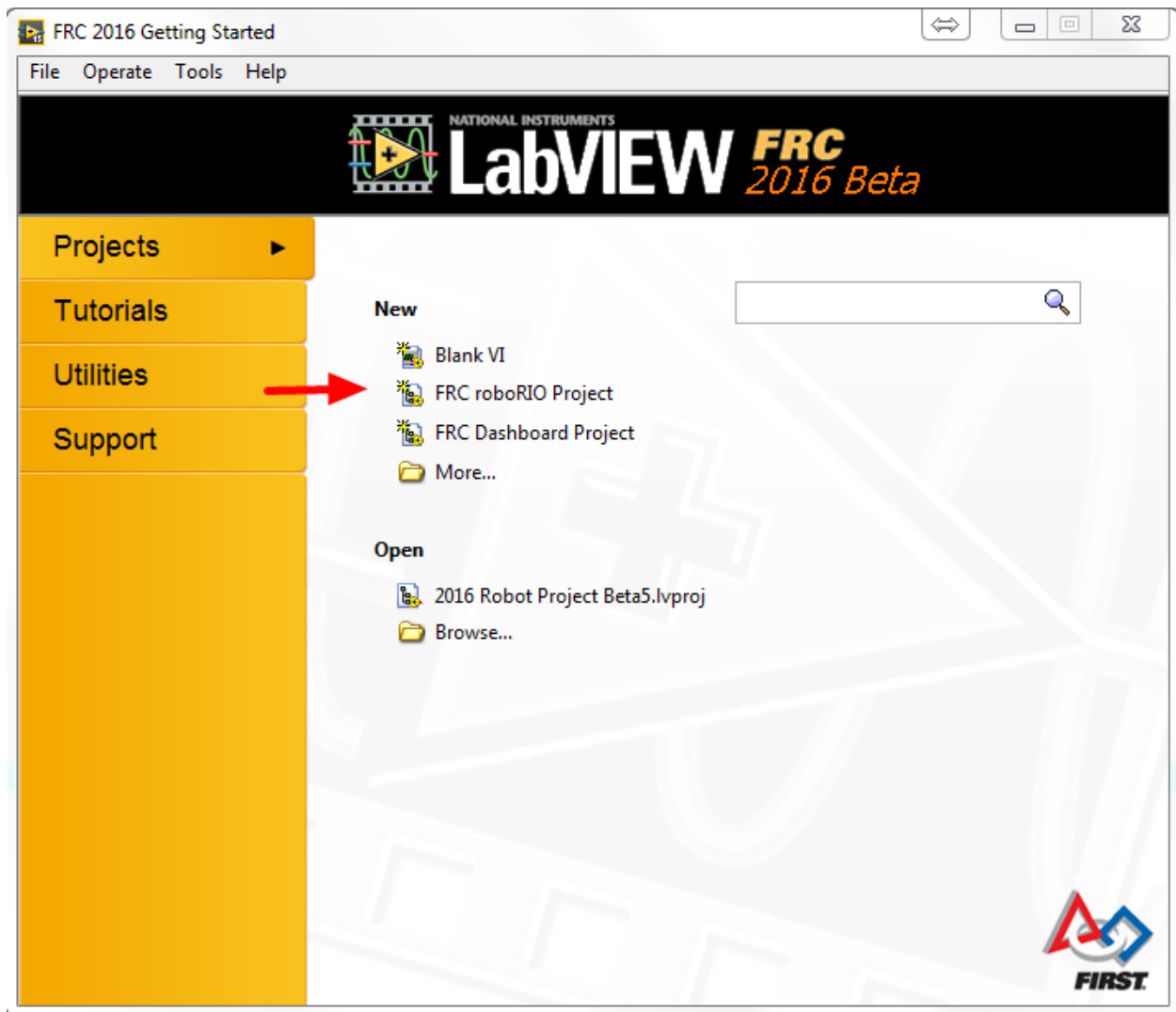
3.1.1 Creating, Building and Loading your Benchtop Test Program



Note: This document covers how to create, build and load an FRC LabVIEW program onto a roboRIO. Before beginning, make sure that you have installed LabVIEW for FRC and the FRC Driver Station and that you have configured and imaged your roboRIO as described in the Getting Started with the Control System section.

Creating a Project

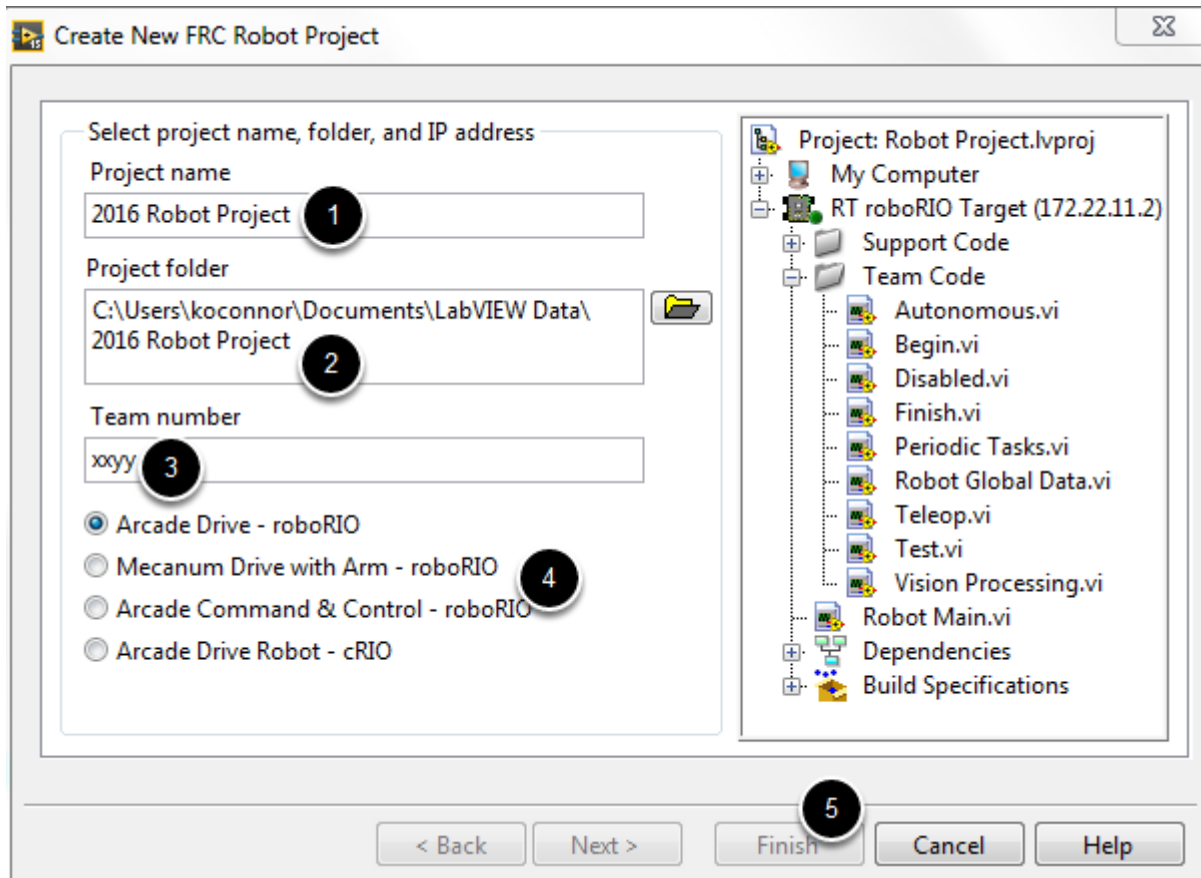
Launch LabVIEW and click the FRC roboRIO Robot Project link in the Projects window to display the Create New FRC Robot Project dialog box.



Configuring Project

Fill in the Create New FRC Project Dialog:

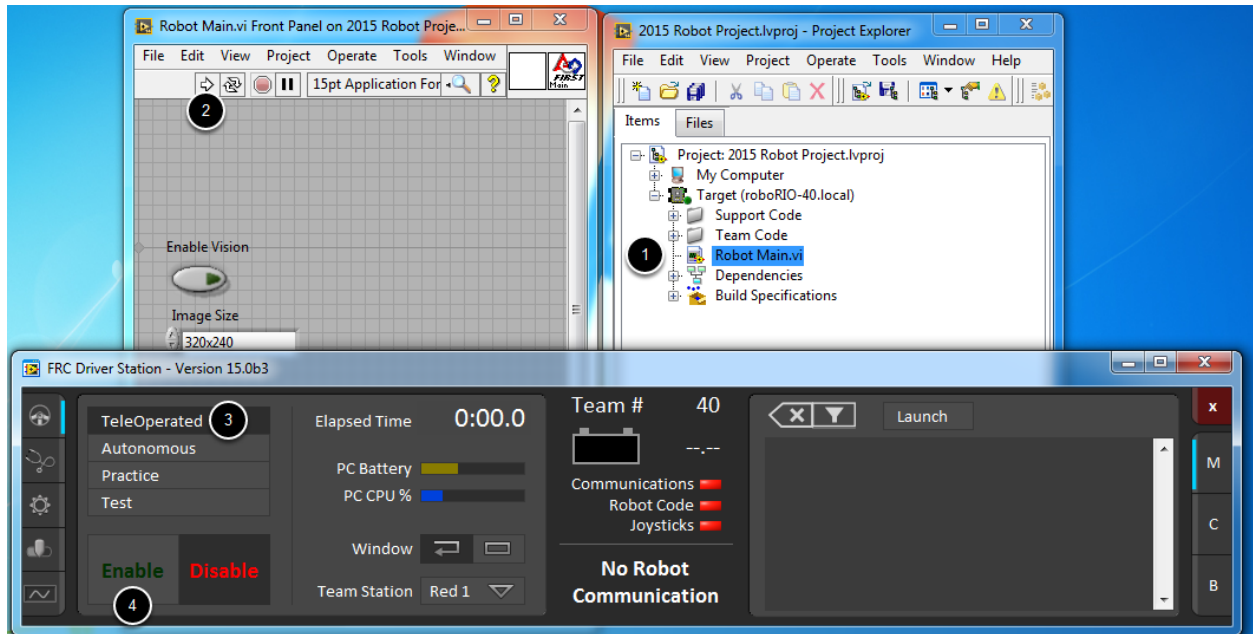
1. Pick a name for your project
2. Select a folder to place the project in.
3. Enter your team number
4. Select a project type. If unsure, select Arcade Drive - roboRIO.
5. Click Finish



Running the Program

Note: Note that a program deployed in this manner will not remain on the roboRIO after a power cycle. To deploy a program to run every time the roboRIO starts follow the next step, Deploying the program.

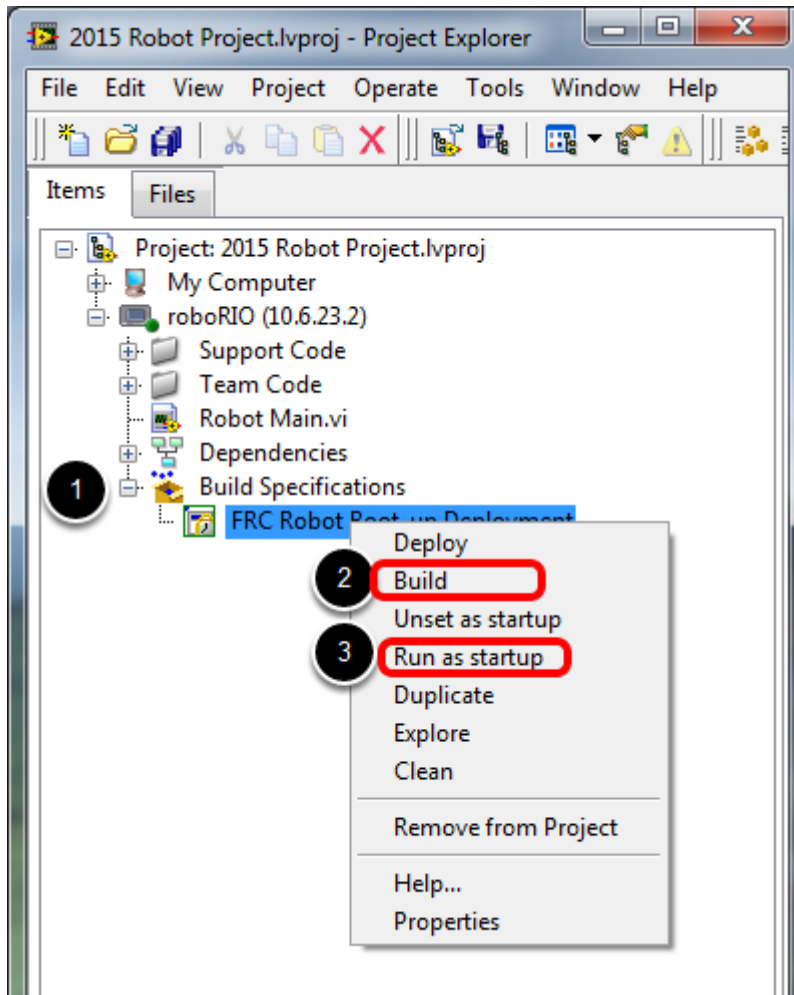
1. In the Project Explorer window, double-click the Robot Main.vi item to open the Robot Main VI.
2. Click the Run button (White Arrow on the top ribbon) of the Robot Main VI to deploy the VI to the roboRIO. LabVIEW deploys the VI, all items required by the VI, and the target settings to memory on the roboRIO. If prompted to save any VIs, click Save on all prompts.
3. Using the Driver Station software, put the robot in Teleop Mode. For more information on configuring and using the Driver Station software, see the FRC Driver Station Software article.
4. Click Enable.
5. Move the joysticks and observe how the robot responds.
6. Click the Abort button of the Robot Main VI. Notice that the VI stops. When you deploy a program with the Run button, the program runs on the roboRIO, but you can manipulate the front panel objects of the program from the host computer.



Deploying the Program

To run in the competition, you will need to deploy a program to your roboRIO. This allows the program to survive across reboots of the controller, but doesn't allow the same debugging features (front panel, probes, highlight execution) as running from the front panel. To deploy your program:

1. In the Project Explorer, click the + next to Build Specifications to expand it.
2. Right-click on FRC Robot Boot-up Deployment and select Build. Wait for the build to complete.
3. Right-click again on FRC Robot Boot-Up Deployment and select Run as Startup. If you receive a conflict dialog, click OK. This dialog simply indicates that there is currently a program on the roboRIO which will be terminated/replaced.
4. Either check the box to close the deployment window on successful completion or click the close button when the deployment completes.
5. The roboRIO will automatically start running the deployed code within a few seconds of the dialog closing.



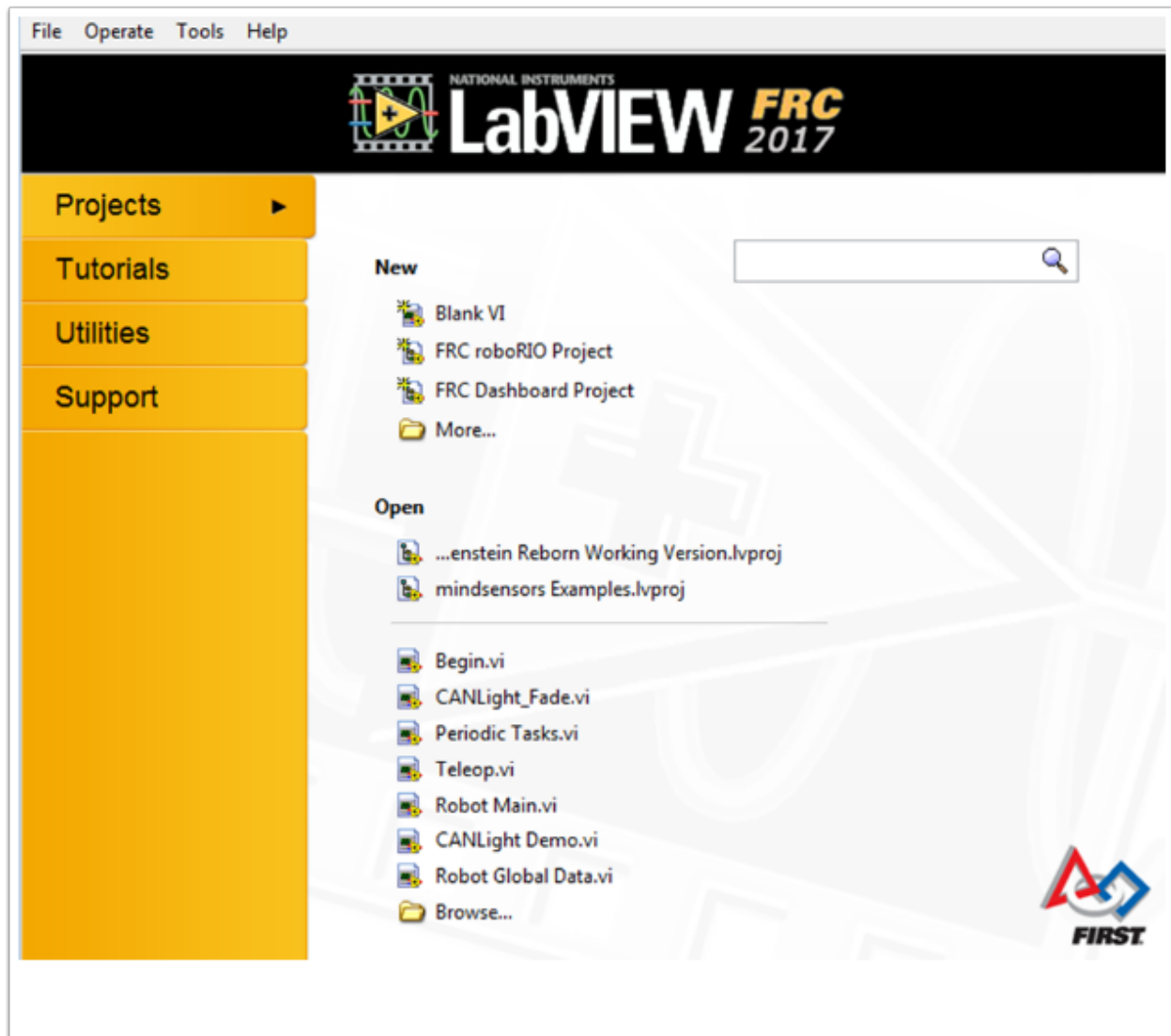
3.1.2 Tank Drive Tutorial



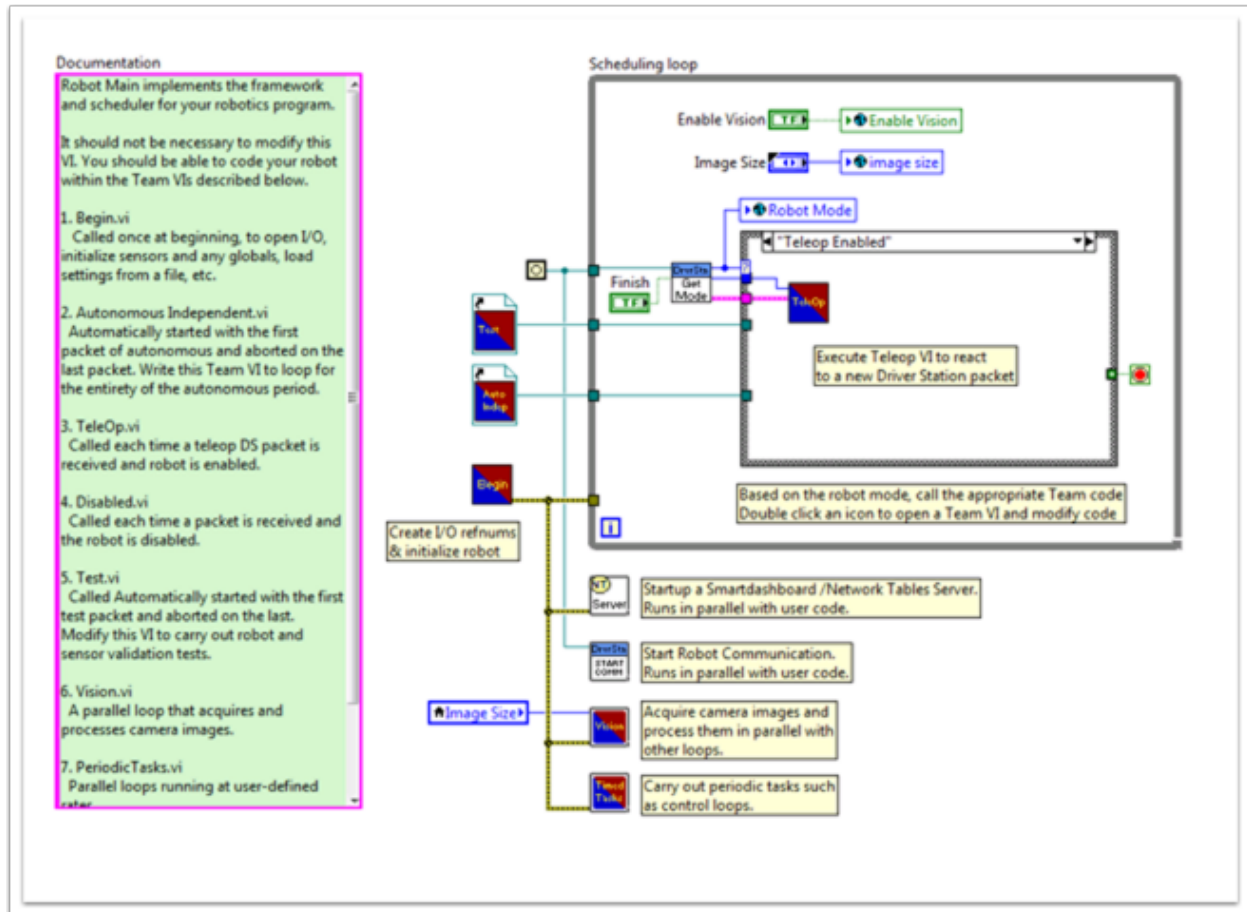
Question: How do I get my robot to drive with two joysticks using tank drive?

Solution: There are four components to consider when setting up tank drive for your robot. The first thing you will want to do is make sure the tank drive.vi is used instead of the arcade drive.vi or whichever drive VI you were utilizing previously. The second item to consider is how you want your joysticks to map to the direction you want to drive. In tank drive, the left joystick is used to control the left motors and the right joystick is used to control the right motors. For example, if you want to make your robot turn right by pushing up on the left joystick and down on the right joystick you will need to set your joystick's accordingly in LabVIEW (this is shown in more detail below). Next, you will want to confirm the PWM lines that you are wired into, are the same ones your joysticks will be controlling. Lastly, make sure your motor controllers match the motor controllers specified in LabVIEW. The steps below will discuss these ideas in more detail:

1. Open LabVIEW and double click FRC roboRIO Project.

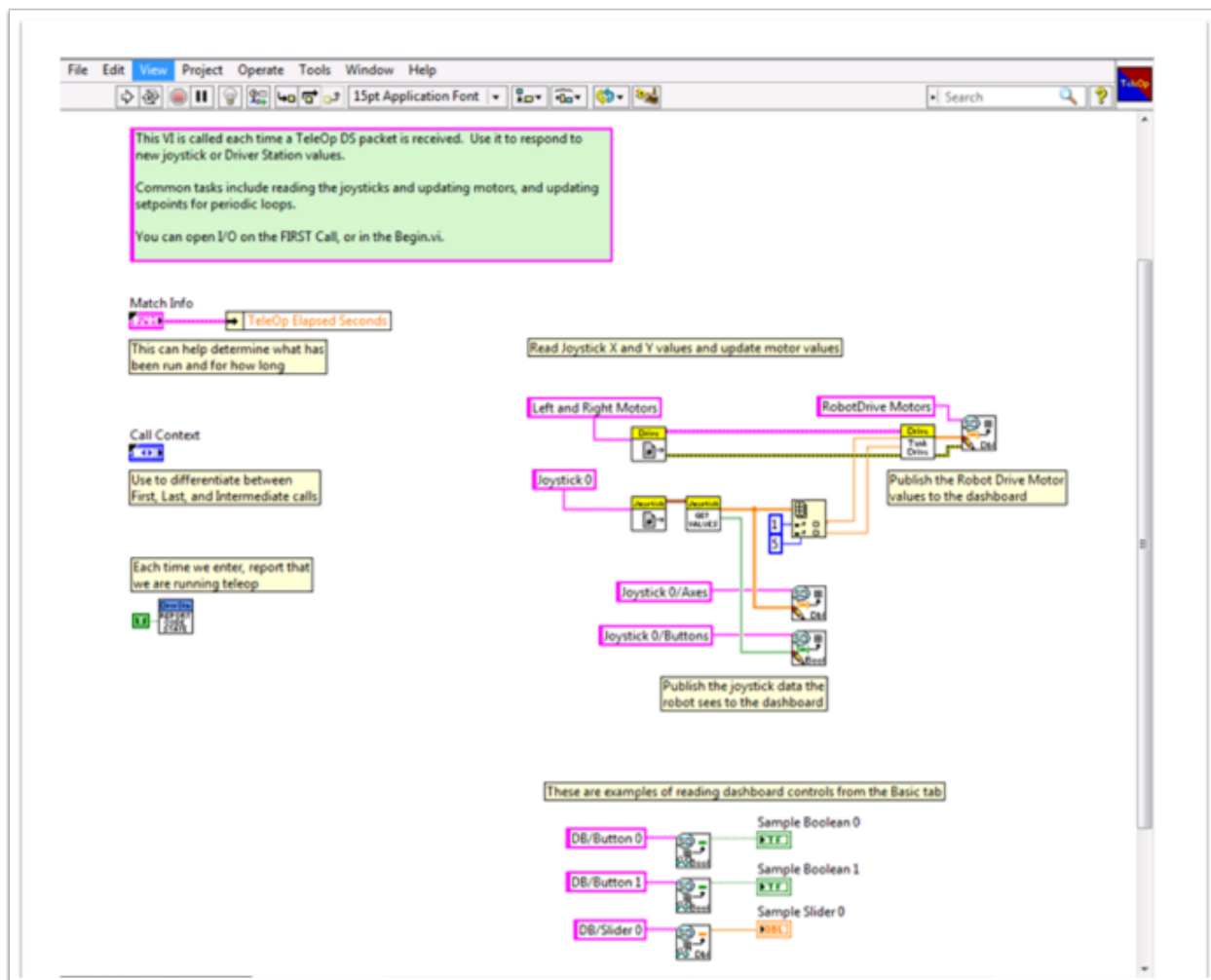


2. Give your project a name, add your team number, and select Arcade Drive Robot roboRIO. You can select another option, however, this tutorial will discuss how to setup tank drive for this project.
3. In the Project Explorer window, open up the Robot Main.vi.
4. Push Ctrl + E to see the block diagram. It should look like the following image:



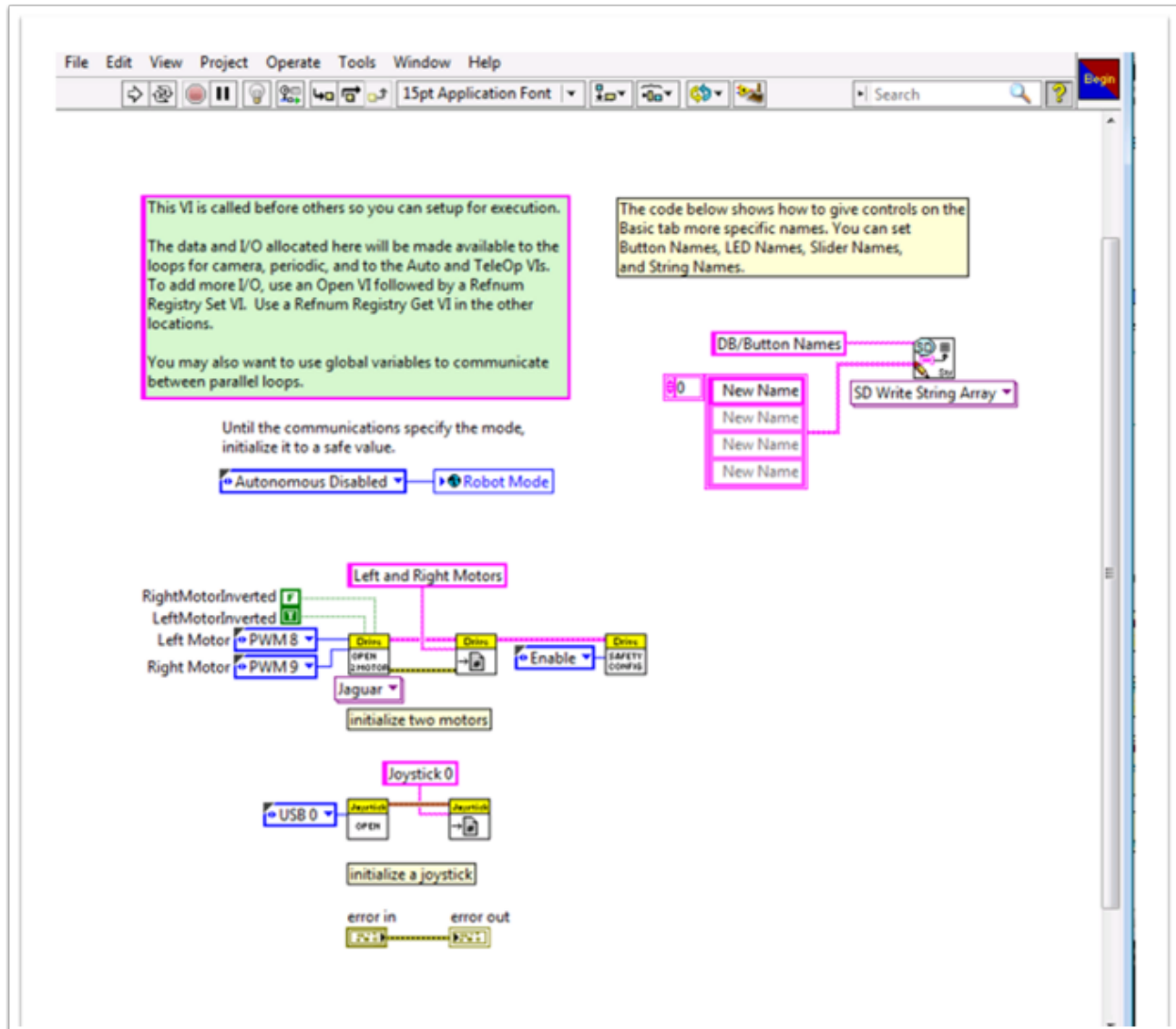
5. Double click the “Teleop” vi inside of the Teleop Enabled case structure. Look at its block diagram. You will want to make two changes here:

- Replace Arcade Drive with the tank drive.vi. This can be found by right clicking on the block diagram >> WPI Robotics Library >> Robot Drive >> and clicking the Tank Drive VI.
- Find the Index Array function that is after the Get Values.vi. You will need to create two numeric constants and wire each into one of the index inputs. You can determine what the values of each index should be by looking at the USB Devices tab in the FRC Driver Station. Move the two joysticks to determine which number (index) they are tied to. You will likely want to use the Y-axis index for each joystick. This is because it is intuitive to push up on the joystick when you want the motors to go forward, and down when you when them to go in reverse. If you select the X-axis index for each, then you will have to move the joystick left or right (x-axis directions) to get the robot motors to move. In my setup, I’ve selected index 1 for my left motors Y-axis control and index 5 as the right motors Y-axis control. You can see the adjustments in LabVIEW in the following image:



6. Next you will want to go back to your “Robot Main.vi” and double click on the “Begin.vi.”
7. The first thing to confirm in this VI is that your left and right motors are connected to the same PWM lines in LabVIEW as they are on your PDP (Power Distribution Panel).
8. The second thing to confirm in this VI is that the “Open 2 Motor.vi” has the correct motor controller selected (Talon, Jaguar, Victor, etc.).

For example, I am using Jaguar motor controllers and my motors are wired into PWM 8 and 9. The image below shows the changes I need to make:



9. Save all of the Vis that you have made adjustments to and you are now able to drive a robot with tank drive!

3.1.3 Command and Control Tutorial



Introduction

Command and Control is a new LabVIEW template added for the 2016 season which organizes robot code into commands and controllers for a collection of robot-specific subsystems. Each subsystem has an independent control loop or state machine running at the appropriate rate

for the mechanism and high-level commands that update desired operations and set points. This makes it very easy for autonomous code to build synchronous sequences of commands. Meanwhile, TeleOp benefits because it can use the same commands without needing to wait for completion, allowing for easy cancellation and initiation of new commands according to the drive team input. Each subsystem has a panel displaying its sensor and control values over time, and command tracing to aid in debugging.

What is Command and Control?

Command and Control recognizes that FRC robots tend to be built up of relatively independent mechanisms such as Drive, Shooter, Arm, etc. Each of these is referred to as a subsystem and needs code that will coordinate the various sensors and actuators of the subsystem in order to complete requested commands, or actions, such as “Close Gripper” or “Lower Arm”. One of the key principles of this framework is that subsystems will each have an independent controller loop that is solely responsible for updating motors and other actuators. Code outside of the subsystem controller can issue commands which may change the robot’s output, but should not directly change any outputs. The difference is very subtle but this means that outputs can only possibly be updated from one location in the project. This speeds up debugging a robot behaving unexpectedly by giving you the ability to look through a list of commands sent to the subsystem rather than searching your project for where an output may have been modified. It also becomes easier to add an additional sensor, change gearing, or disable a mechanism without needing to modify code outside of the controller.

Game code, primarily consisting of Autonomous and TeleOp, will typically need to update set points and react to the state of certain mechanisms. For Autonomous, it is very common to define the robot’s operation as a sequence of operations – drive here, pick that up, carry it there, shoot it, etc. Commands can be wired sequentially with additional logic to quickly build complex routines. For teleOp, the same commands can execute asynchronously, allowing the robot to always process the latest driver inputs, and if implemented properly, new commands will interrupt, allowing the drive team to quickly respond to field conditions while also taking advantage of automated commands and command sequences.

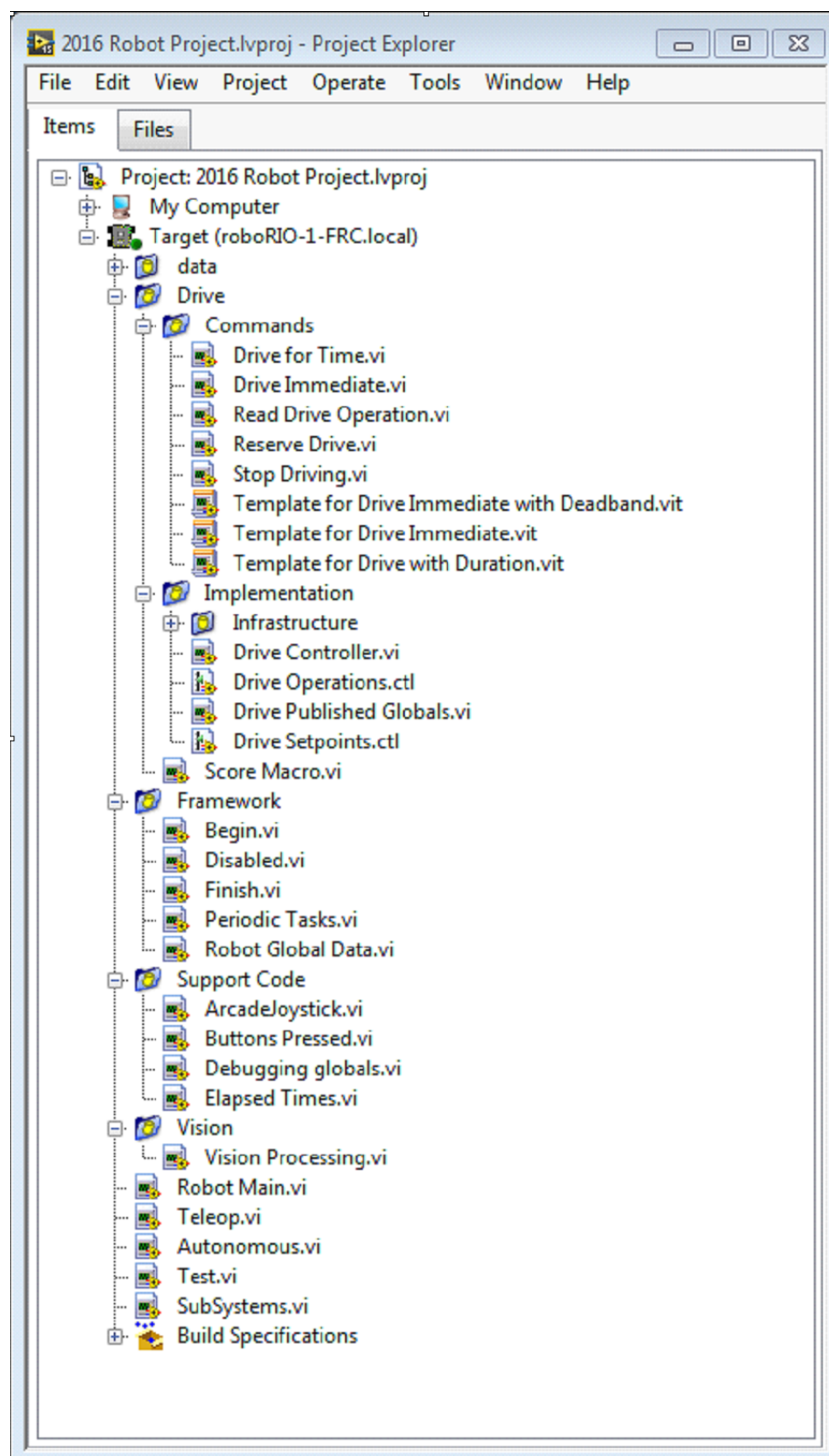
Why should I use Command and Control?

Command and Control adds functionality to the existing LabVIEW project templates, allowing code to scale better with more sophisticated robots and robot code. Subsystems are used to abstract the details of the implementation, and game code is built from sequences of high level command VIs. The commands themselves are VIs that can update set points, perform numerical scaling/mapping between engineering units and mechanism units, and offer synchronization options. If physical changes are made to the robot, such as changing a gearing ratio, changes can be made to just a few command VIs to reflect this change across the entire code base.

I/O encapsulation makes for more predictable operation and quicker debugging when resource conflicts do occur. Because each command is a VI, you are able to single step through commands or use the built in Trace functionality to view a list of all commands sent to each subsystem. The framework uses asynchronous notification and consistent data propagation making it easy to program a sequence of commands or add in simple logic to determine the correct command to run.

Part 1: Project Explorer

The Project Explorer provides organization for all of the Vis and files you will use for your robot system. Below is a description of the major components in the Project Explorer to help with the expansion of our system. The most frequently used items have been bolded.



My Computer The items that define operation on the computer that the project was loaded on. For a robot project, this is used as a simulation target and is populated with simulation files.

Sim Support Files The folder containing 3D CAD models and description files for the simulated robot.

Robot Simulation Readme.html Documents the PWM channels and robot info you will need in order to write robot code that matches the wiring of the simulated robot.

Dependencies Shows the files used by the simulated robot's code. This will populate when you designate the code for the simulated robot target.

Build Specifications This will contain the files that define how to build and deploy code for the simulated robot target.

Target (roboRIO-TEAM-FRC.local) The items that define operation on the roboRIO located at (address).

Drive The subsystem implementation and commands for the robot drive base. This serves as a custom replacement for the WPILib RobotDrive VIs.

Framework VIs used for robot code that is not part of a subsystem that are not used very often.

Begin Called once when robot code first starts. This is useful for initialization code that doesn't belong to a particular subsystem.

Disabled Called once for each disabled packet and can be used to debug sensors when you don't want the robot to move.

Finish During development, this may be called when robot code finishes. Not called on abort or when power is turned off.

Periodic Tasks A good place for ad hoc periodic loops for debugging or monitoring

Robot Global Data Useful for sharing robot information that doesn't belong to a subsystem.

Support Code Debugging and code development aids.

Vision Subsystem and commands for the camera and image processing.

Robot Main.vi Top level VI that you will run while developing code.

Autonomous.vi VI that runs during autonomous period.

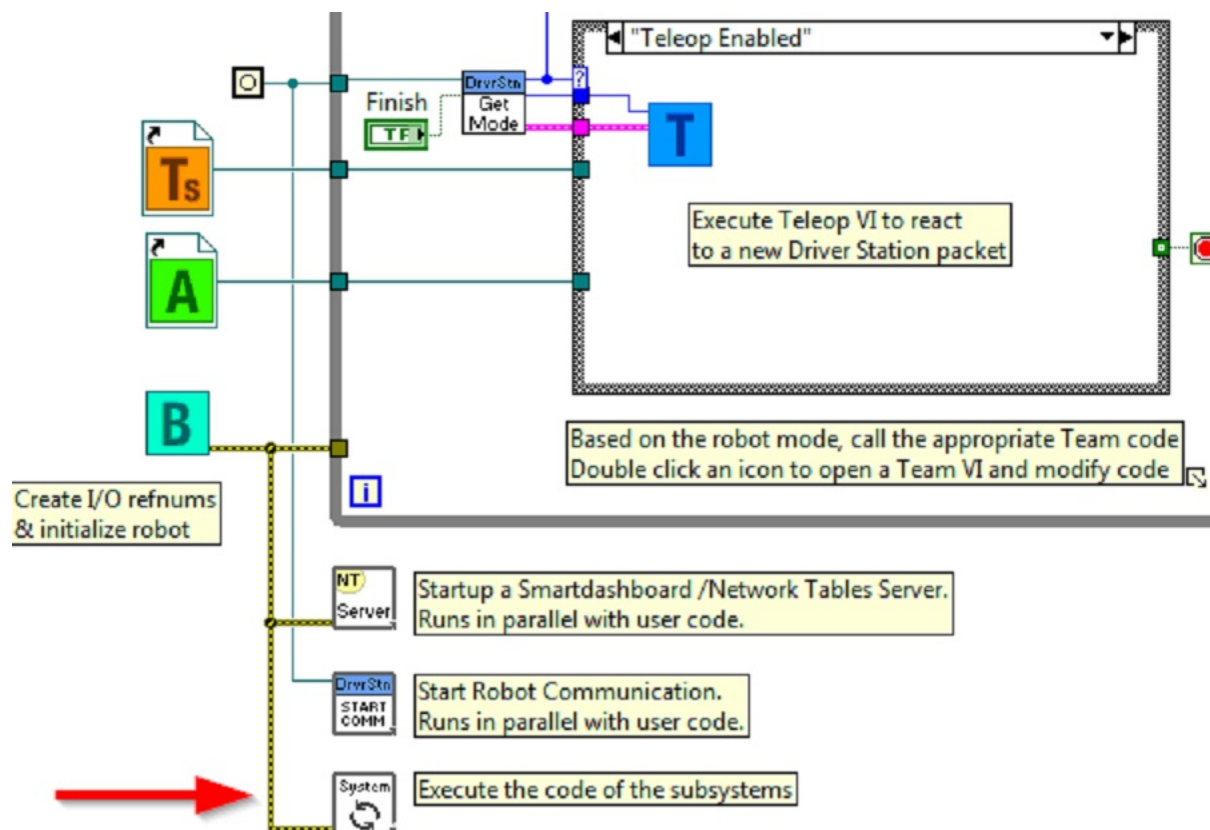
Teleop.vi VI that is called for each TeleOp packet.

Test.vi VI that runs when driver station is in test mode.

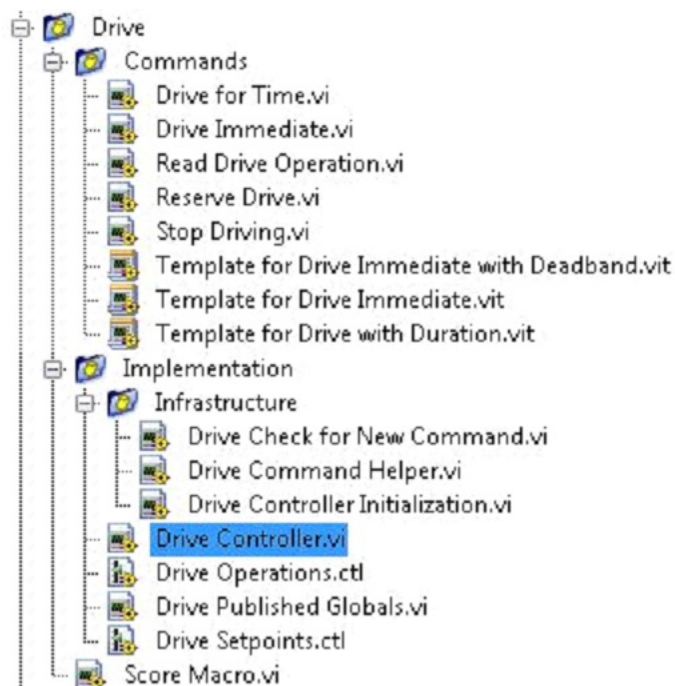
SubSystems.vi VI that contains and starts all subsystems.

Dependencies Shows the files used by the robot code.

Build Specifications Used to build and run the code as a startup application once code works correctly.



Drive Subsystem Project Explorer



Commands: This folder contains the command VIs that request the controller carry out an

operation. It also contains templates for creating additional drive commands.

Note: After creating a new command, you may need to edit `Drive Setpoints.ctl` to add or update fields that controller uses to define the new operation. You also need to go into the `Drive Controller.vi` and modify the case structure to add a case for every value.

Implementation

These are the VIs and Controls used to build the subsystem.

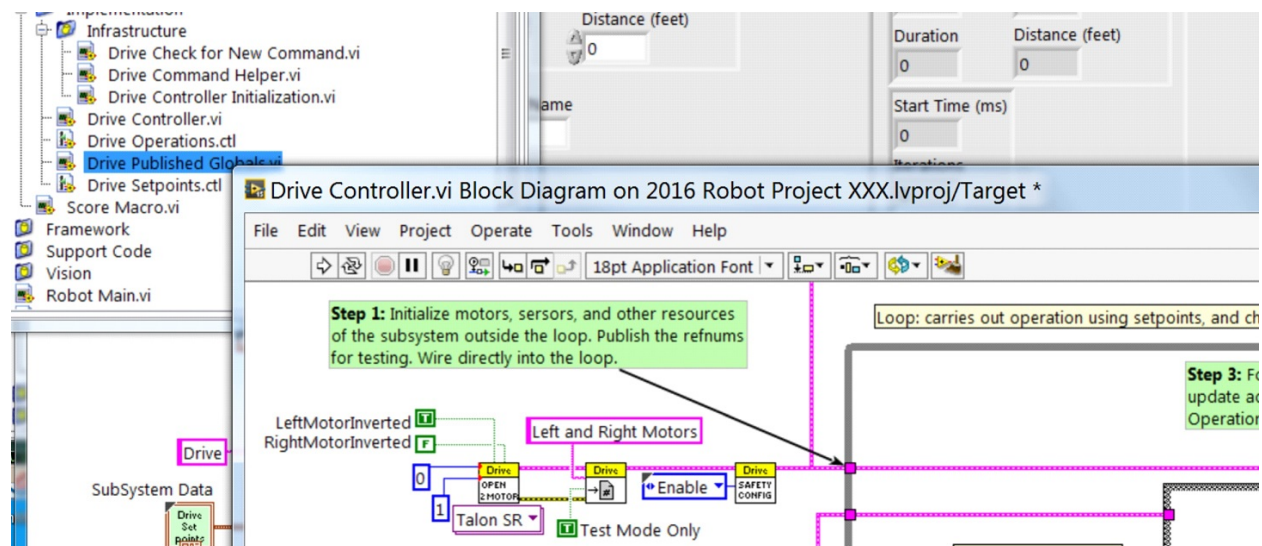
Infrastructure VIs

- **Drive Check for New Command:** It is called each iteration of the controller loop. It checks for new commands, updates timing data, and upon completion notifies a waiting command.
- **Drive Command Helper.vi:** Commands call this VI to notify the controller that a new command has been issued.
- **Drive Controller Initialization.vi:** It allocates the notifier and combines the timing, default command, and other information into a single data wire.
- **Drive Controller.vi:** This VI contains the control/state machine loop. The panel may also contain displays useful for debugging.
- **Drive Operation.ctl:** This typedef defines the operational modes of the controller. Many commands can share an operation.
- **Drive Setpoint.ctl:** It contains the data fields used by all operating modes of the Drive subsystem.
- **Drive Published Globals.vi:** A useful place for publishing global information about the drive subsystem.

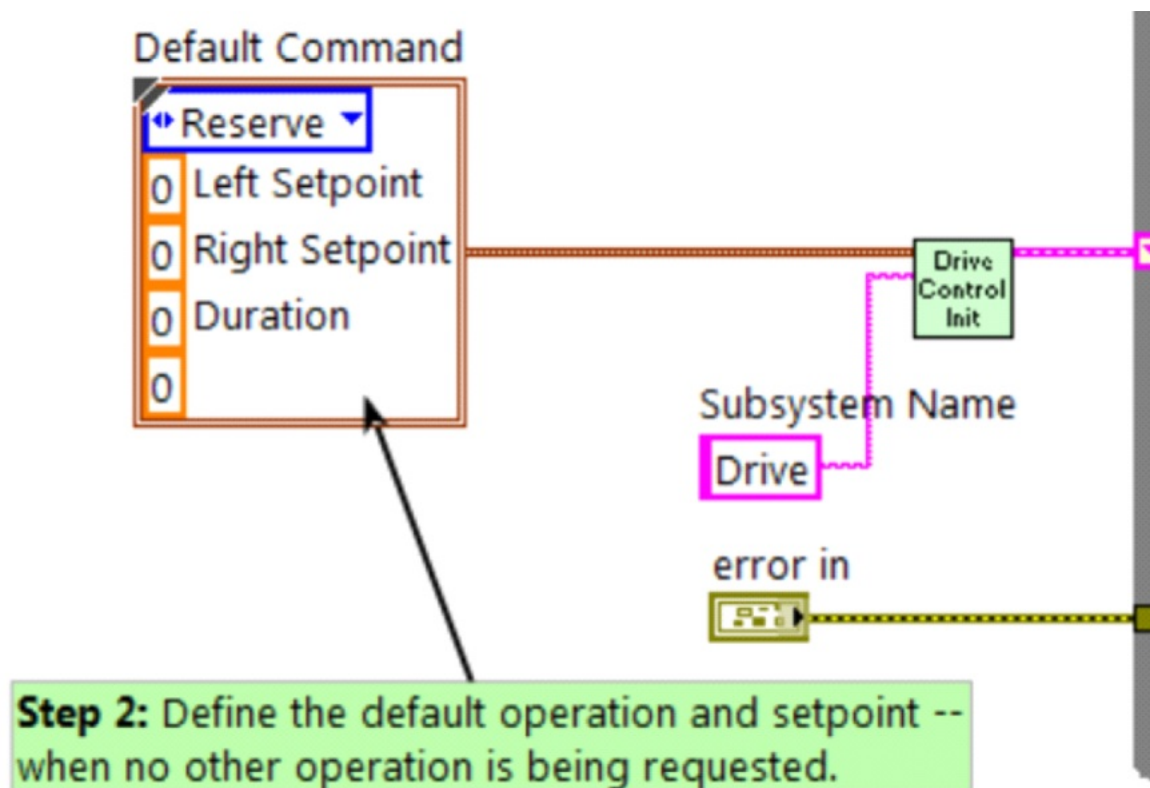
Part 2: Initializing the Drive Subsystem

There are green comments on the controller's block diagram that point out key areas that you will want to know how to edit.

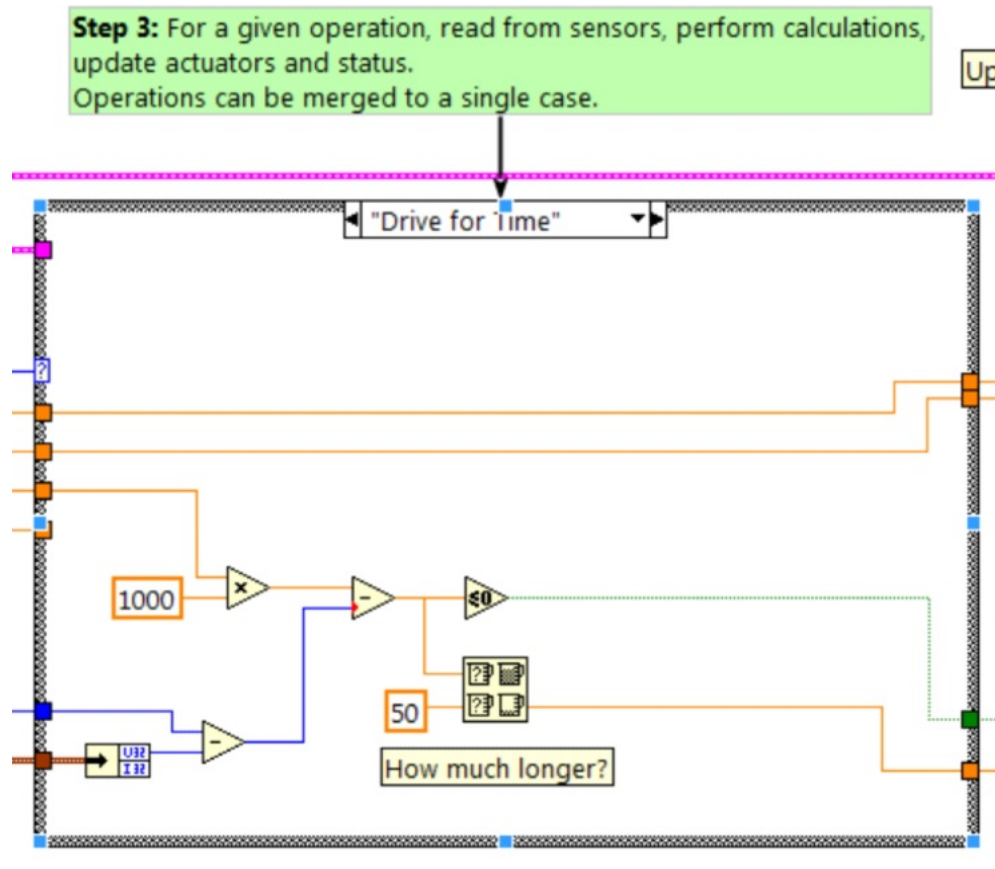
The area to the left of the control loop will execute once when the subsystem starts up. This is where you will typically allocate and initialize all I/O and state data. You may publish the I/O refs, or you may register them for Test Mode Only to keep them private so that other code cannot update motors without using a command.



Note: Initializing the resources for each subsystem in their respective Controller.vi rather than in Begin.vi improves I/O encapsulation, reducing potential resource conflicts and simplifies debugging.

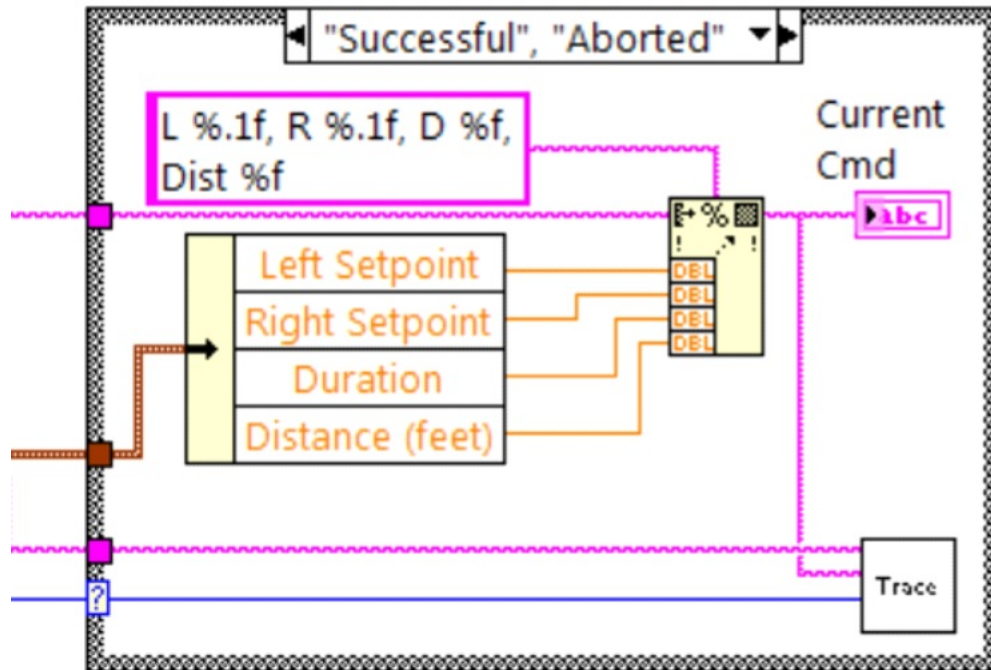


Part of the initialization is to select the default operation and set point values when no other operation is being processed.



Inside the control loop is a case statement where operations are actually implemented. Set point values, iteration delay, iteration count, and sensors can all have influence on how the subsystem operates. This case structure has a value for each operation state of the subsystem.

Step 4: Format a string to describe this cmd and how the previous cmd finished



Each iteration of the controller loop will optionally update the Trace VI. The framework already incorporates the subsystem name, operation, and description, and you may find it helpful to format additional set point values into the trace information. Open the Trace VI and click Enable while the robot code is running to current setpoints and commands sent to each subsystem.

The primary goal of the controller is to update actuators for the subsystem. This can occur within the case structure, but many times, it is beneficial to do it downstream of the structure to ensure that values are always updated with the correct value and in only one location in the code.

Step 5: Update your I/O

Update motors and update dashboard

RobotDrive Motors



Part 3: Drive Subsystem Shipped Commands

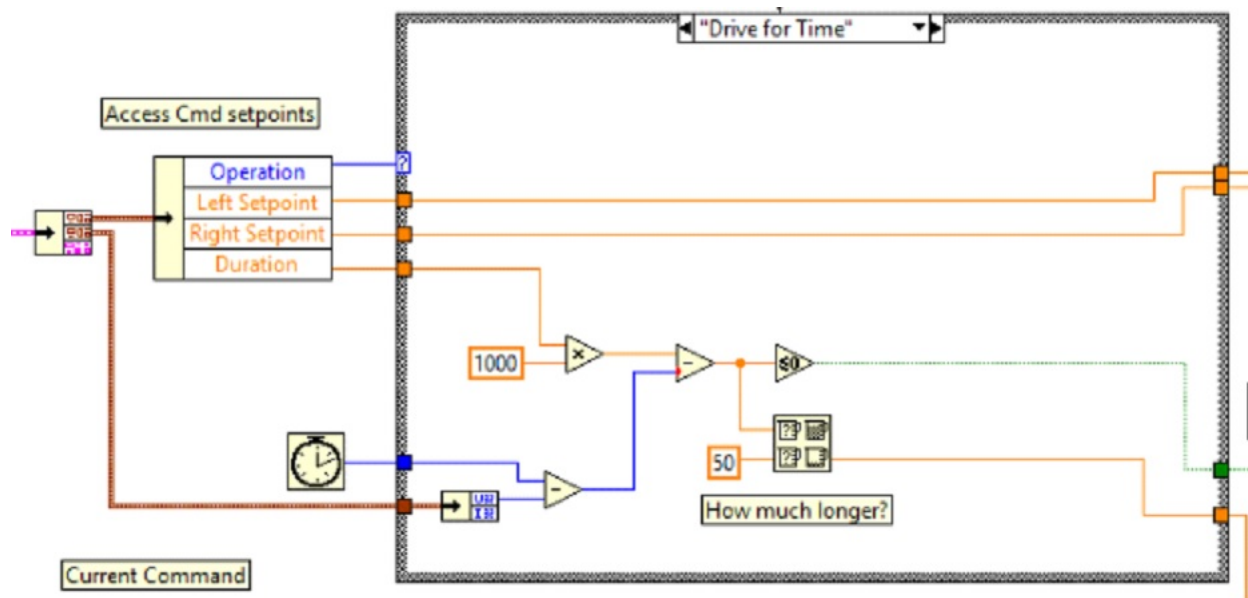
There are 3 shipped example commands for each new subsystem:

Drive For Time.vi



This VI sets the motors to run for a given number of seconds. It optionally synchronizes with the completion of the command.

The Drive for Time case will operate the motors at the set point until the timer elapses or a new command is issued. If the motors have the safety timeout enabled, it is necessary to update the motors at least once every 100ms. This is why the code waits for the smaller of the remaining time and 50ms.

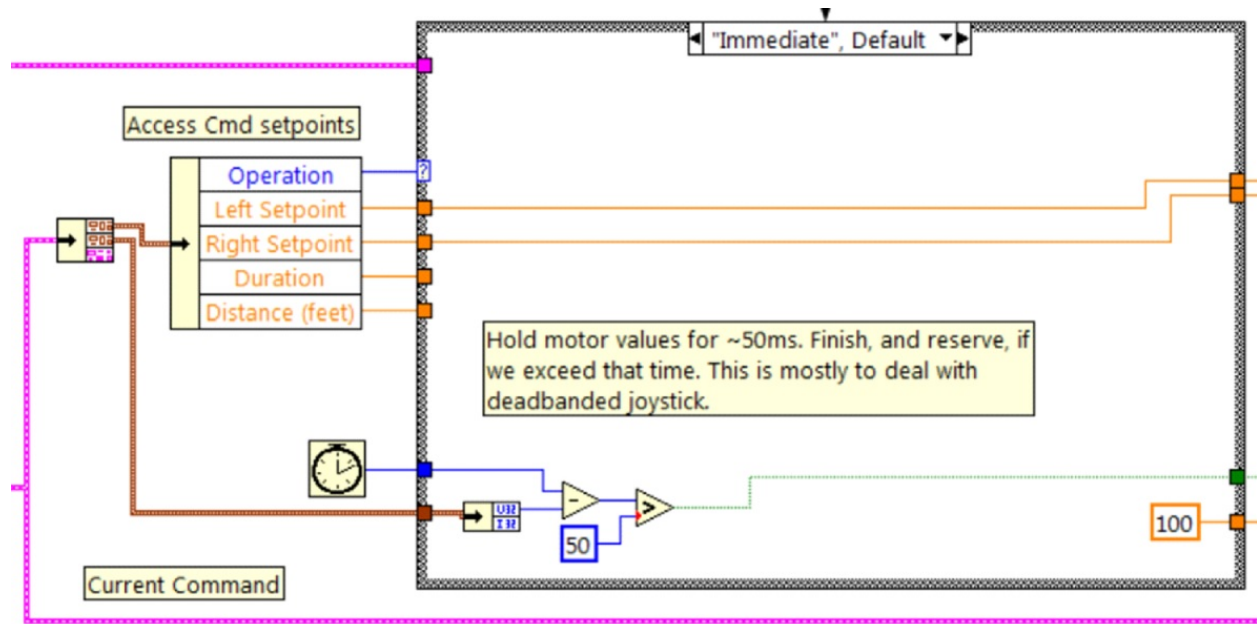


Drive Immediate.vi



Gets the desired left and right speeds for the motors and will set the motors immediately to those set points.

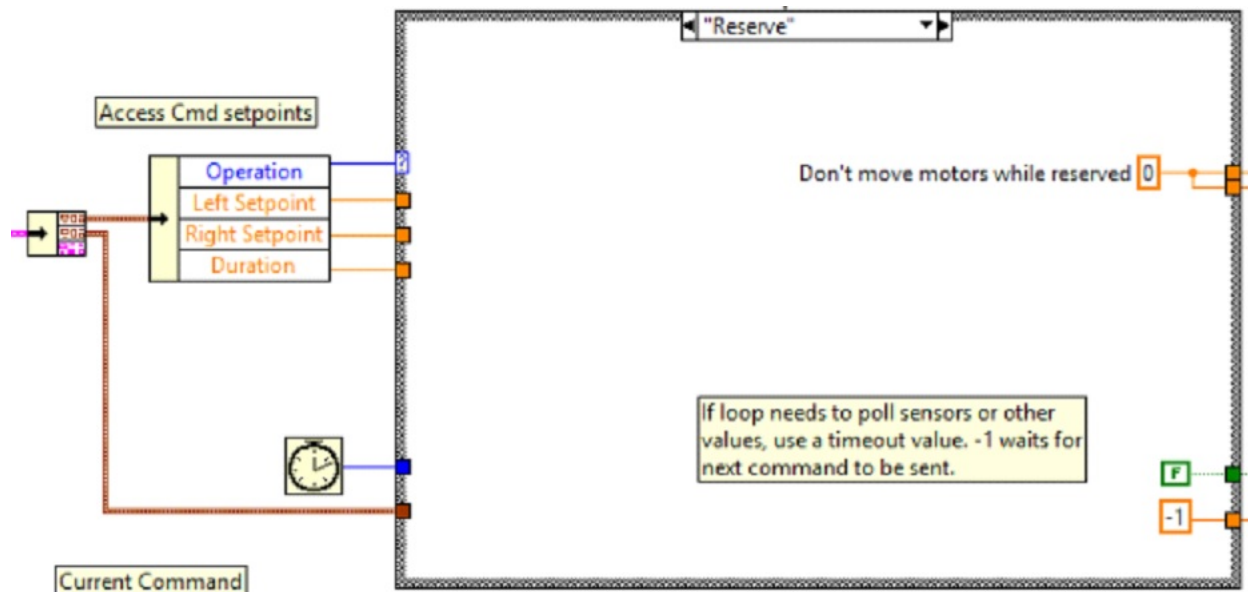
The Immediate case updates the motors to the set point defined by the command. The command is not considered finished since you want the motors to maintain this value until a new command comes in or until a timeout value. The timeout is useful anytime a command includes a dead band. Small values will not be requested if smaller than the dead band, and will result in growling or creeping unless the command times out.



Stop Driving.vi

Zero the drive motors, making the robot stationary.

The Reserve command turns off the motors and waits for a new command. When used with a named command sequence, reserve identifies that the drive subsystem is part of a sequence, even if not currently moving the robot. This helps to arbitrate subsystem resource between simultaneously running commands.

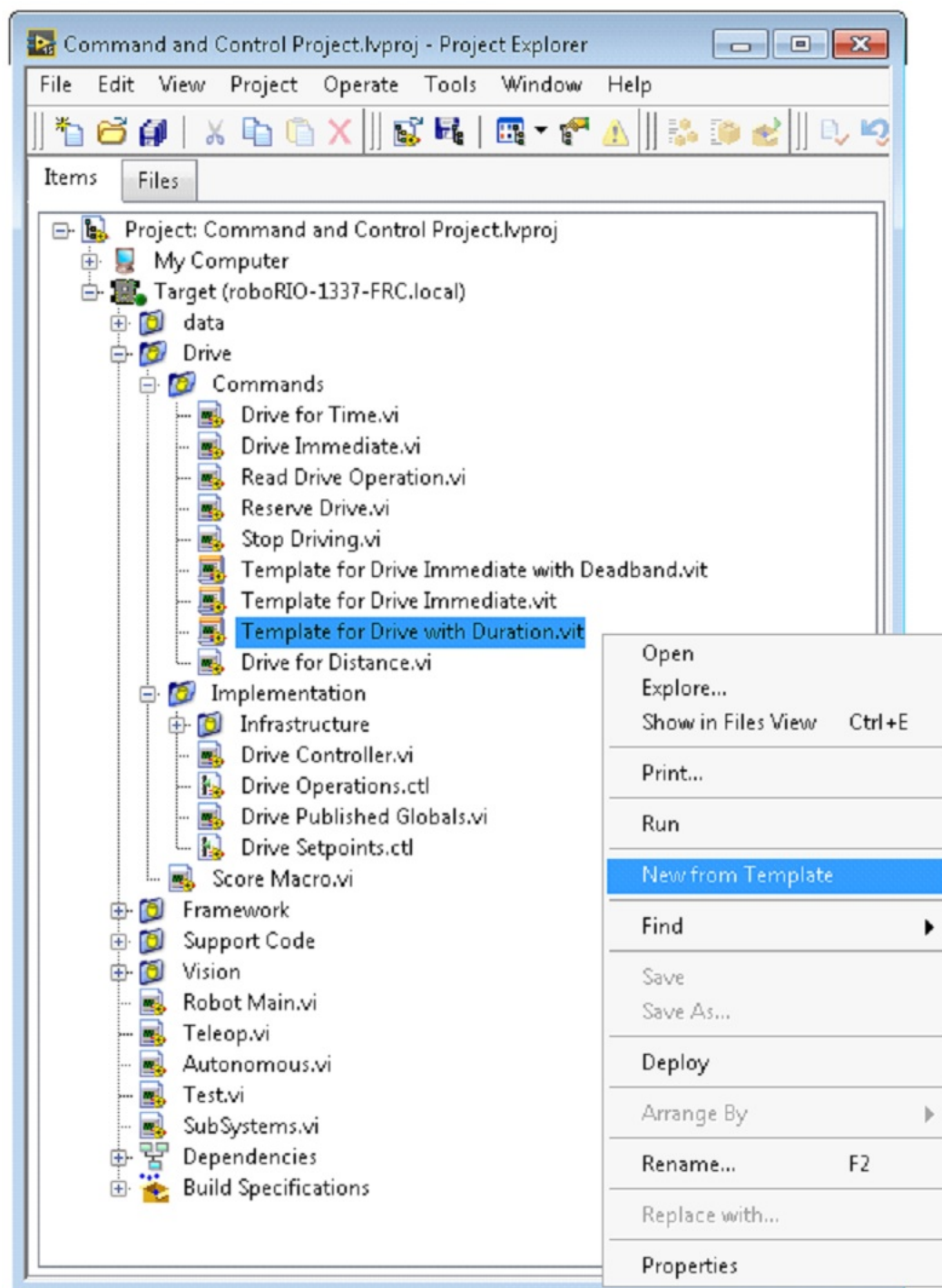


Part 4: Creating New Commands

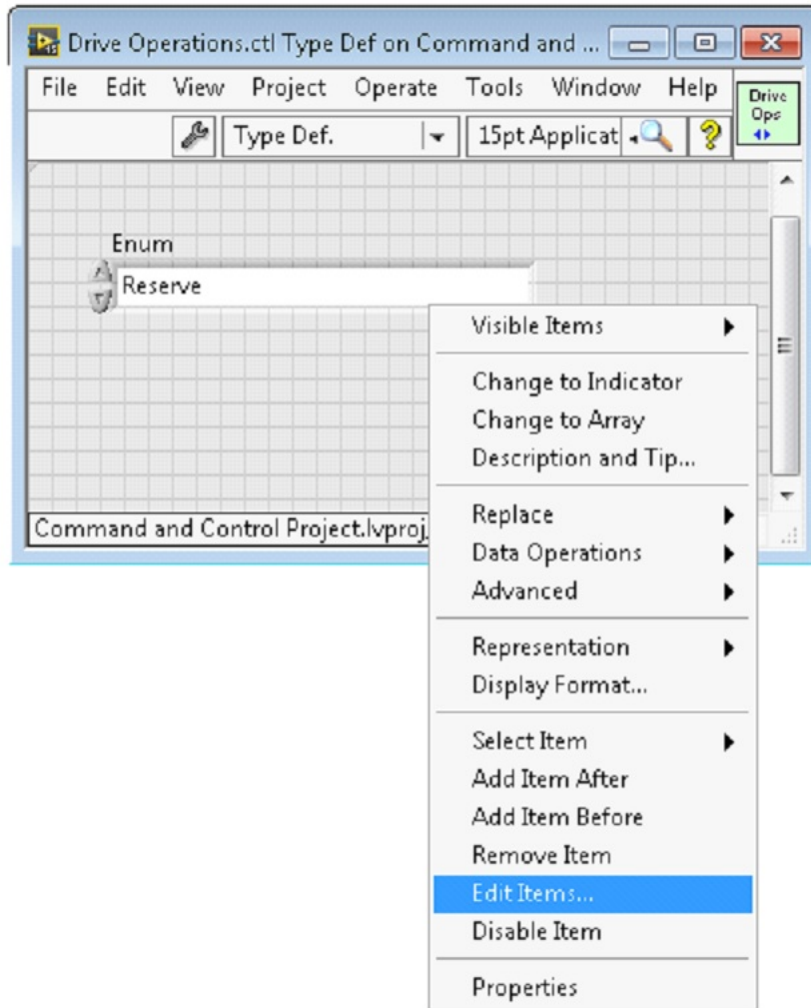
The Command and Control framework allows users to easily create new commands for a subsystem. To Create a new command open the subsystem folder/Commands In the project explorer window, choose one of the VI Templates to use as the starting point of your new command, right click, and select New From Template.

- **Immediate:** This VI notifies the subsystem about the new setpoint.
- **Immediate with deadband:** This VI compares the input value to the deadband and optionally notifies the subsystem about the new setpoint. This is very useful when joystick continuous values are being used.
- **With duration:** This VI notifies the subsystem to perform this command for the given duration, and then return to the default state. Synchronization determines whether this VI Starts the operation and returns immediately, or waits for the operation to complete. The first option is commonly used for TeleOp, and the second for Autonomous sequencing.

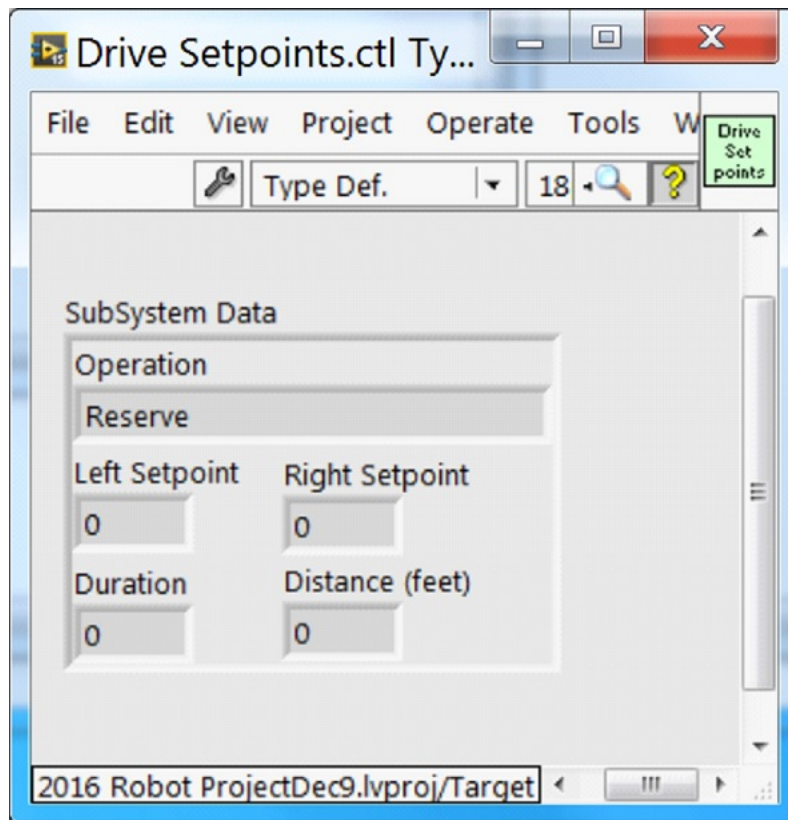
In this example we will add the new command “Drive for Distance”.



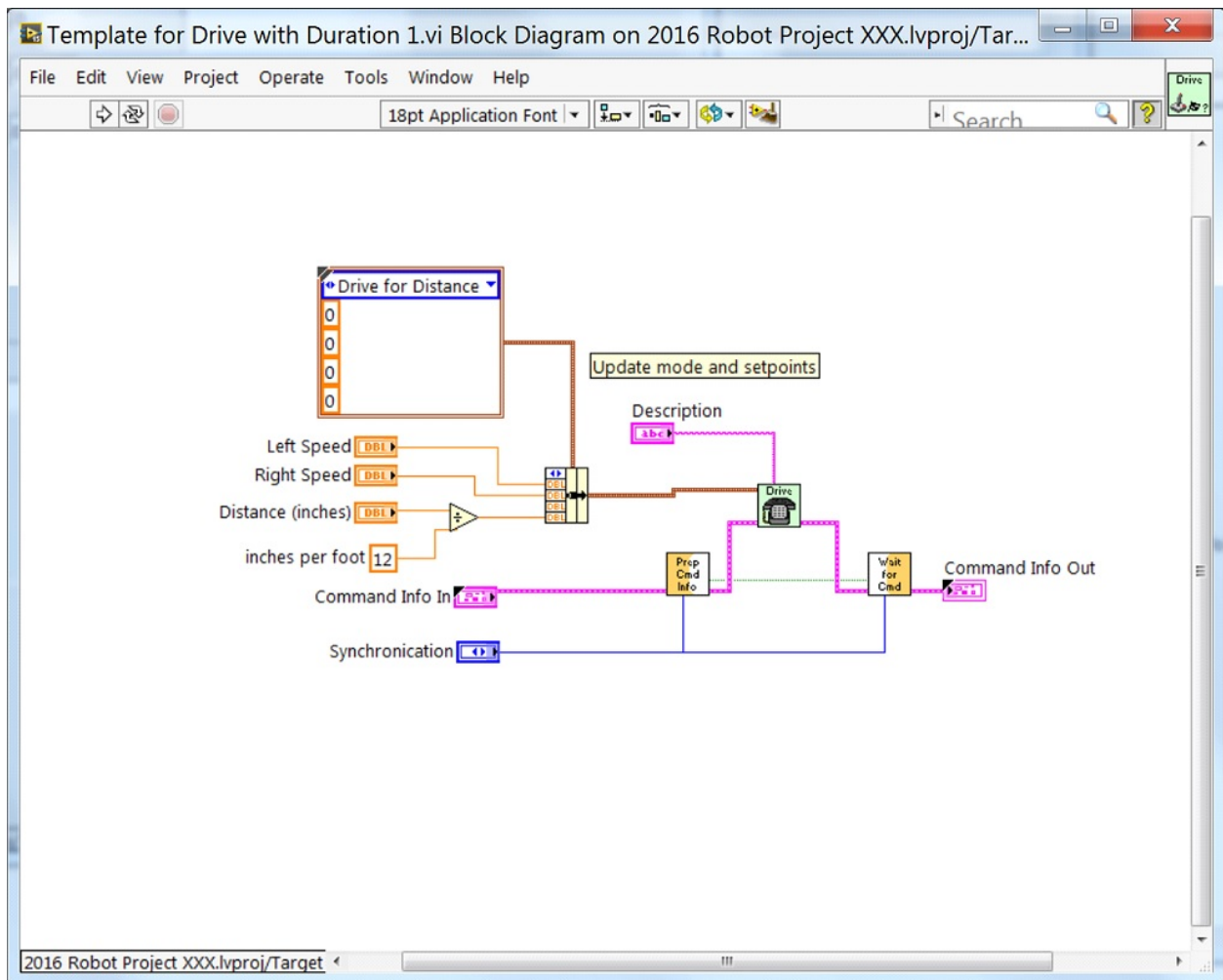
First, save the new VI with a descriptive name such as “Drive for Distance”. Next, determine whether the new command needs a new value added the Drive Operations enum typedef. The initial project code already has an enum value of Drive for Distance, but the following image shows how you would add one if needed.



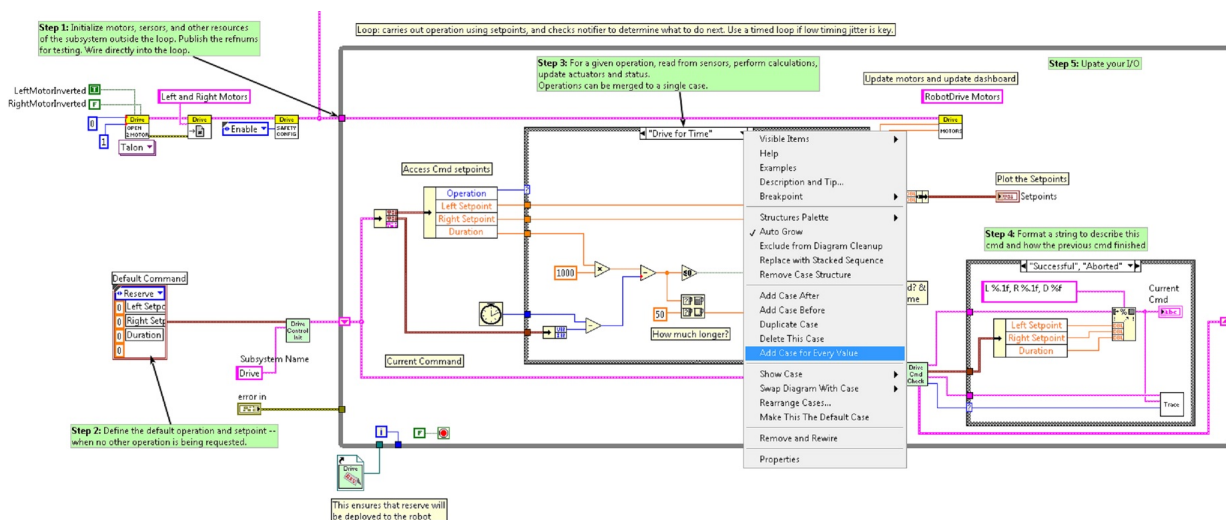
If a command needs additional information to execute, add it to the setpoints control. By default, the Drive subsystem has fields for the Left Setpoint, Right Setpoint, and Duration along with the operation to be executed. The Drive for Distance command could reuse Duration as distance, but let's go ahead and add a numeric control to the Drive Setpoints.ctl called Distance (feet).



Once that we have all of the fields needed to specify our command, we can modify the newly created Drive for Distance.vi. As shown below, select Drive for Distance from the enum's drop down menu and add a VI parameters to specify distance, speeds, etc. If the units do not match, the command VI is a great place to map between units.

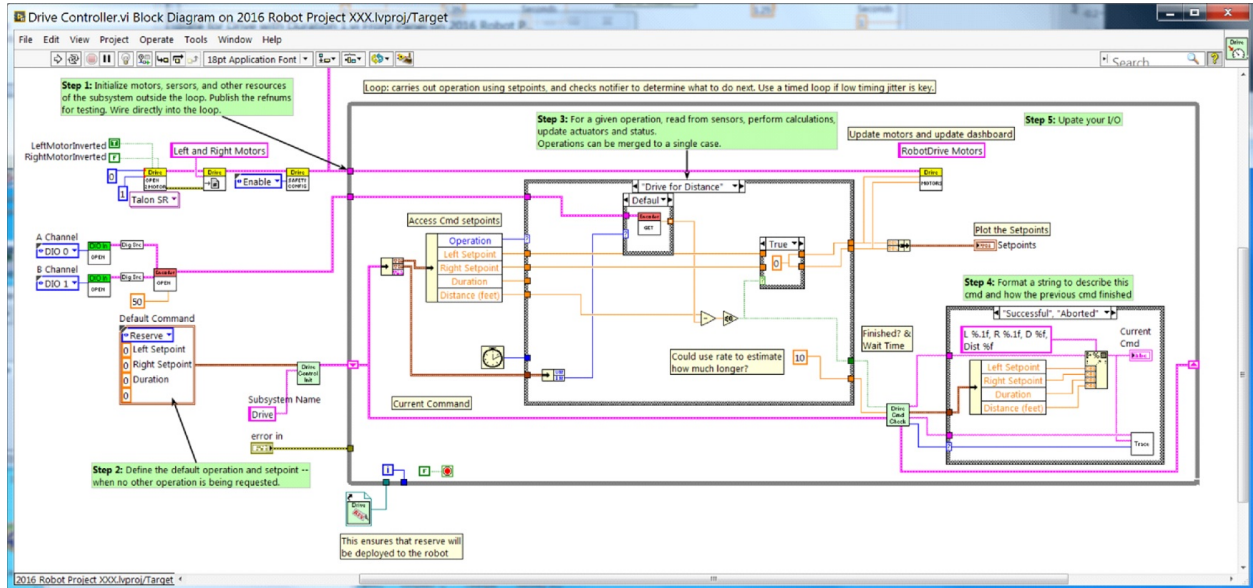


Next, add code to the Drive Controller to define what happens when the Drive for Distance command executes. Right click on the Case Structure and Duplicate or Add Case for Every Value. This will create a new “Drive for Distance” case.



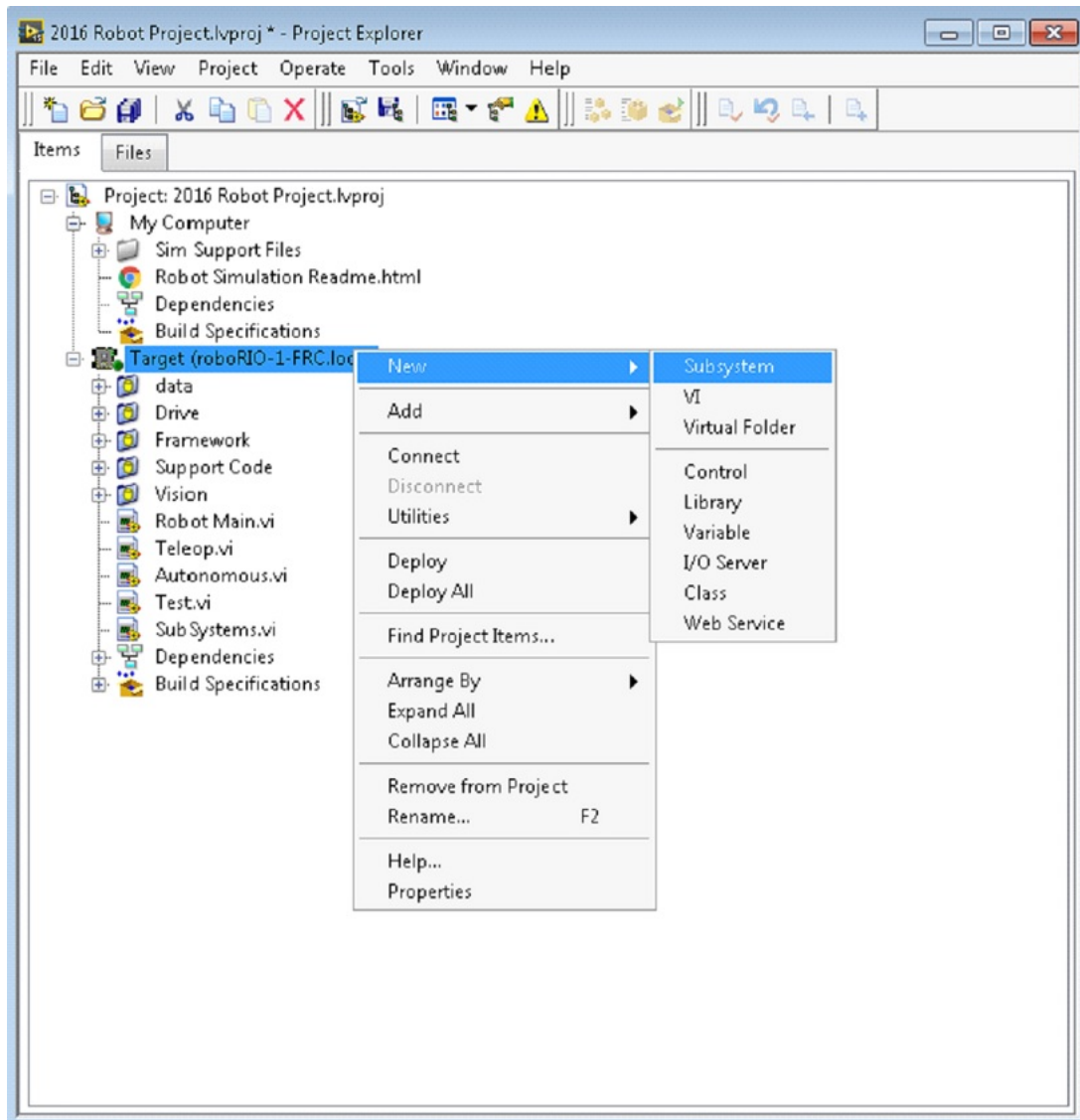
In order to access new setpoint fields, grow the “Access Cmd setpoints” unbundle node. Open

your encoder(s) on the outside, to the left of the loop. In the new diagram of the case structure, we added a call to reset the encoder on the first loop iteration and read it otherwise. There is also some simple code that compares encoder values and updates the motor power. If new controls are added to the setpoints cluster, you should also consider adding them to the Trace. The necessary changes are shown in the image below.

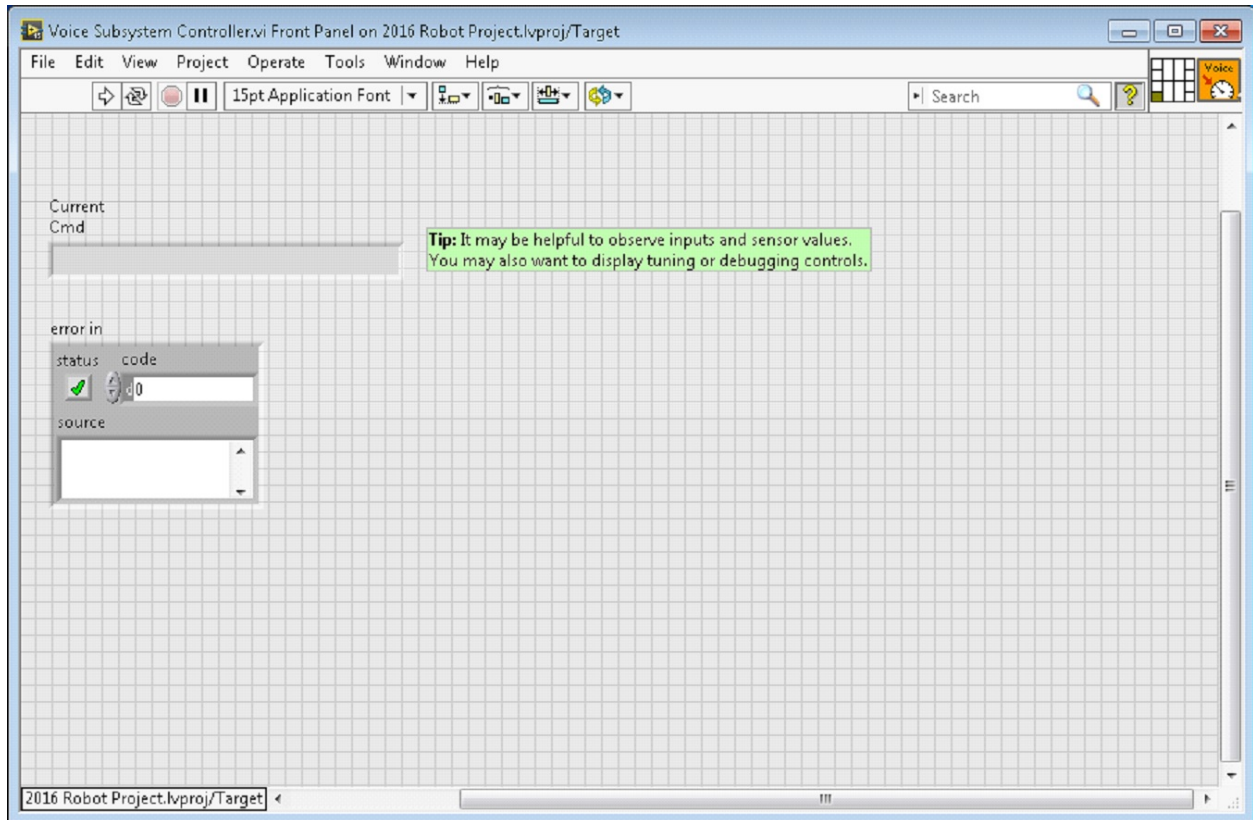


Part 5: Creating a Subsystem

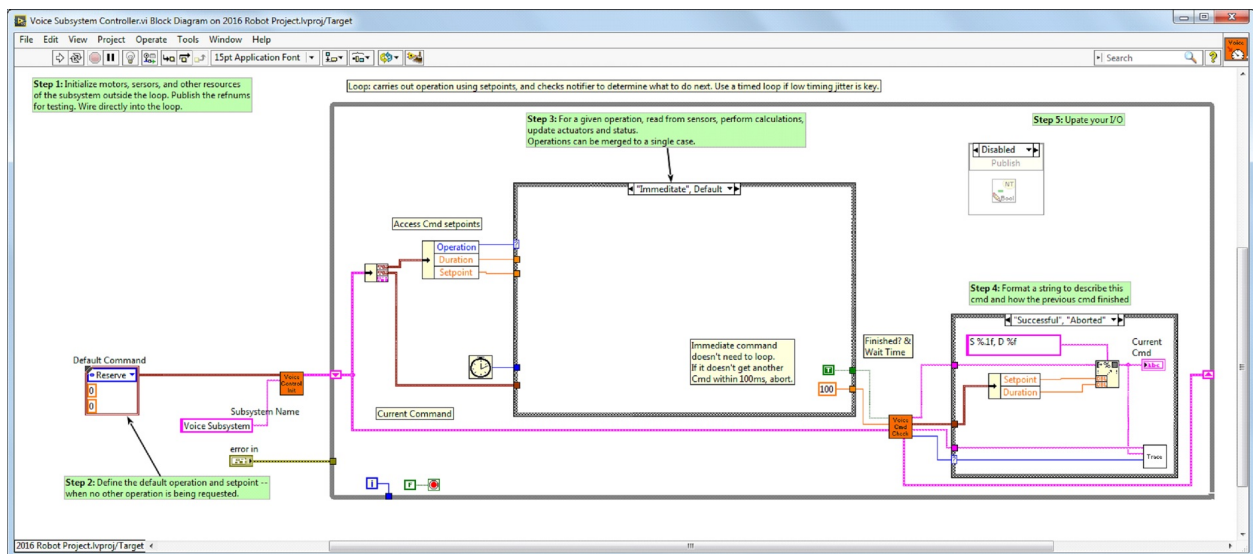
In order to create a new subsystem, right click on the roboRIO target and select New» Subsystem. In the pop up dialog box, enter the name of the subsystem, list the operational modes, and specify the color of the icon.



When you click OK, the subsystem folder will be generated and added to the project disk folder and tree. It will contain a base implementation of the VIs and controls that make up a subsystem. A call to the new controller will be inserted into the Subsystems VI. The controller VI will open, ready for you to add I/O and implement state machine or control code. Generated VI icons will use the color and name provided in the dialog. The generated code will use typedefs for set point fields and operations.



Below is the block diagram of the newly created subsystem. This code will be generated automatically when you create the subsystem.



3.2 LabVIEW Resources

3.2.1 LabVIEW Resources



Note: To learn more about programming in LabVIEW and specifically programming FRC robots in LabVIEW, check out the following resources.

LabVIEW Basics

National Instruments provides a [combination of videos and traditional text/picture tutorials on the basics of LabVIEW](#). These tutorials can help you get acquainted with the LabVIEW environment and the basics of the graphical, dataflow programming model used in LabVIEW.

NI FRC Tutorials

National Instruments also hosts many [FRC specific tutorials and presentations ranging from basic to advanced](#). For an in-depth single resource check out the FRC Basic and Advanced Training Classes linked near the bottom of the page.

Installed Tutorials and Examples

There are also tutorials and examples for all sorts of tasks and components provided as part of your LabVIEW installation. To access the tutorials, from the LabVIEW Splash screen (the screen that appears when the program is first launched) click on the Tutorials tab on the left side. Note that the tutorials are all in one document, so once it is open you are free to browse to other tutorials without returning to the splash screen.

To access the examples either click the Support tab, then Find FRC Examples or anytime you're working on a program open the Help menu, select Find Examples and open the FRC Robotics folder.

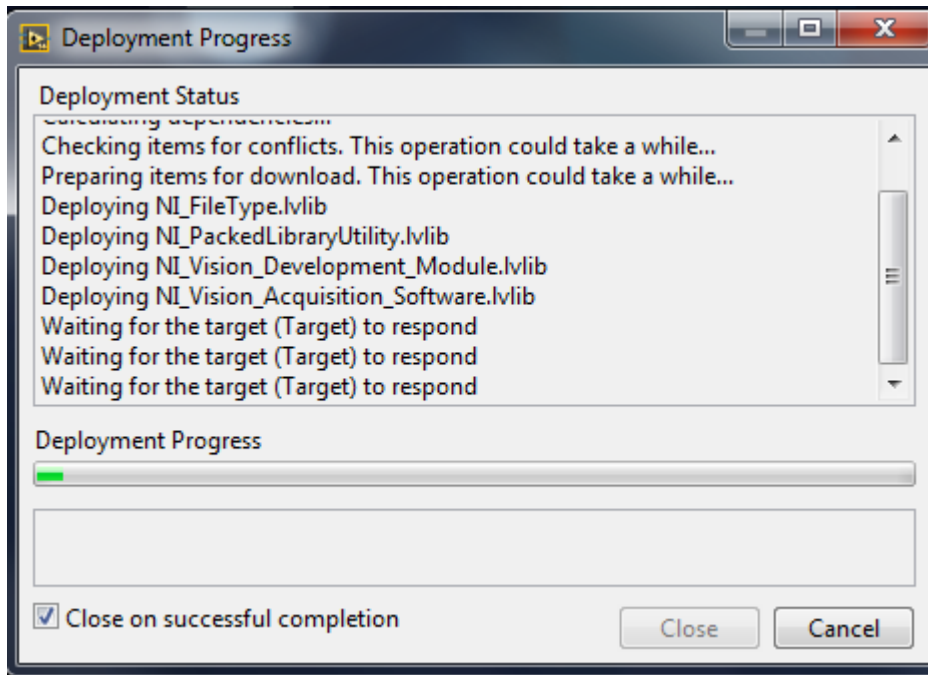
3.2.2 Waiting for Target to Respond - Recovering from bad loops



Note: If you download LabVIEW code which contains an unconstrained loop (a loop with no delay) it is possible to get the roboRIO into a state where LabVIEW is unable to connect to

download new code. This document explains the process required to load new, fixed, code to recover from this state.waiting-for-target-to-respond/

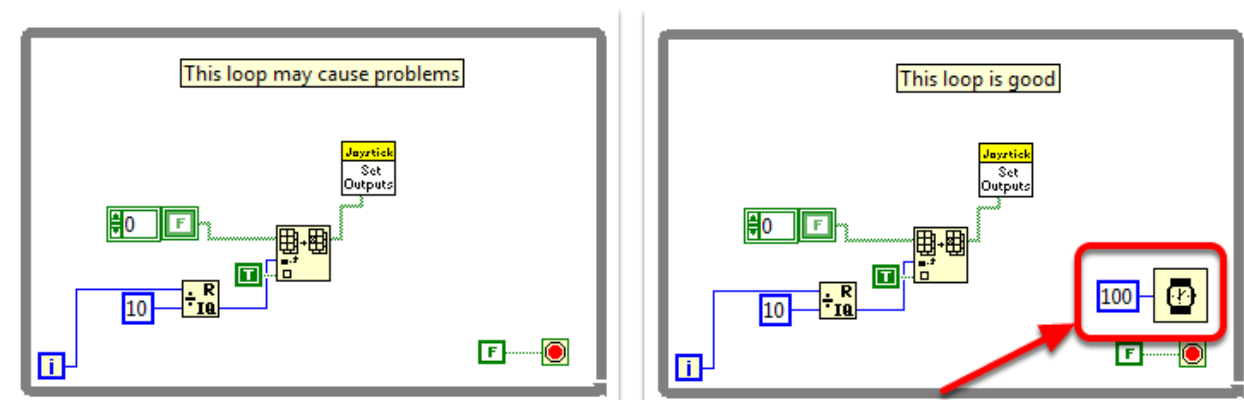
The Symptom



The primary symptom of this issue is attempts to download new robot code hang at the “Waiting for the target (Target) to respond” step as shown above. Note that there are other possible causes of this symptom (such as switching from a C++/Java program to LabVIEW program) but the steps described here should resolve most or all of them.

Click Cancel to close the download dialog.

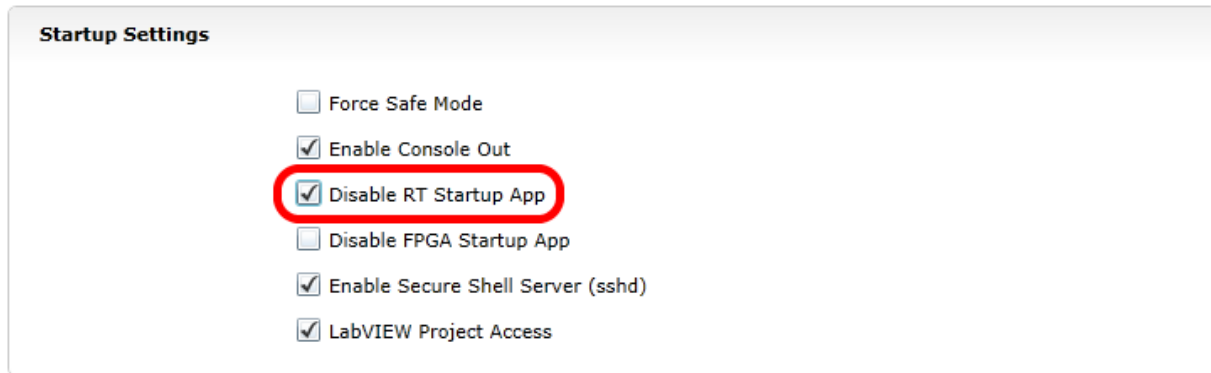
The Problem



One common source of this issue is unconstrained loops in your LabVIEW code. An unconstrained loop is a loop which does not contain any delay element (such as the one on the left).

If you are unsure where to begin looking, Disabled.VI, Periodic Tasks.VI and Vision Processing.VI are the common locations for this type of loop. To fix the issue with the code, add a delay element such as the Wait (ms) VI from the Timing palette, found in the right loop.

Set No App



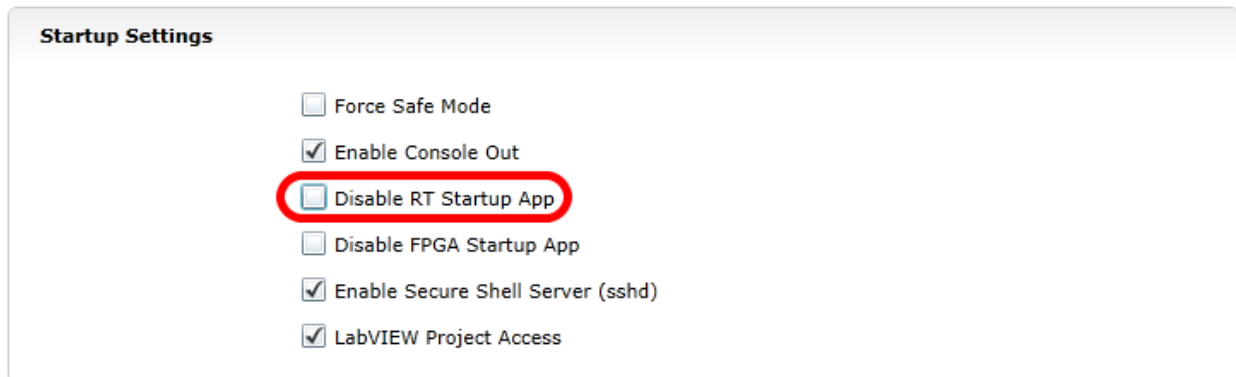
The screenshot shows the 'Startup Settings' web interface. It contains a list of settings with checkboxes. The 'Disable RT Startup App' checkbox is checked and is circled in red. The other settings are: 'Force Safe Mode' (unchecked), 'Enable Console Out' (checked), 'Disable FPGA Startup App' (unchecked), 'Enable Secure Shell Server (sshd)' (checked), and 'LabVIEW Project Access' (checked).

Using the roboRIO webserver (see the article roboRIO Webdashboard for more details). **Check** the box to “Disable RT Startup App”.

Reboot

Reboot the roboRIO, either using the Reset button on the device or by click Restart in the top right corner of the webpage.

Clear No App



The screenshot shows the 'Startup Settings' web interface. It contains a list of settings with checkboxes. The 'Disable RT Startup App' checkbox is unchecked and is circled in red. The other settings are: 'Force Safe Mode' (unchecked), 'Enable Console Out' (checked), 'Disable FPGA Startup App' (unchecked), 'Enable Secure Shell Server (sshd)' (checked), and 'LabVIEW Project Access' (checked).

Using the roboRIO webserver (see the article roboRIO Webdashboard for more details). **Uncheck** the box to “Disable RT Startup App”.

Load LabVIEW Code

Load LabVIEW code (either using the Run button or Run as Startup). Make sure to set LabVIEW code to Run as Startup before rebooting the roboRIO or you will need to follow the

instructions above again.

3.2.3 Talon SRX CAN

The Talon SRX motor controller is a CAN-enabled “smart motor controller” from Cross The Road Electronics/VEX Robotics. The Talon SRX can be controlled over the CAN bus or PWM interface. When using the CAN bus control, this device can take inputs from limit switches and potentiometers, encoders, or similar sensors in order to perform advanced control such as limiting or PID(F) closed loop control on the device.

Extensive documentation about programming the Talon SRX in all three FRC languages can be found in the [Talon SRX Software Reference Manual on CTRE’s Talon SRX product page](#).

Note: CAN Talon SRX has been removed from WPILib. See [this blog](#) for more info and find the CTRE Toolsuite installer [here](#)

3.2.4 How To Toggle Between Two Camera Modes



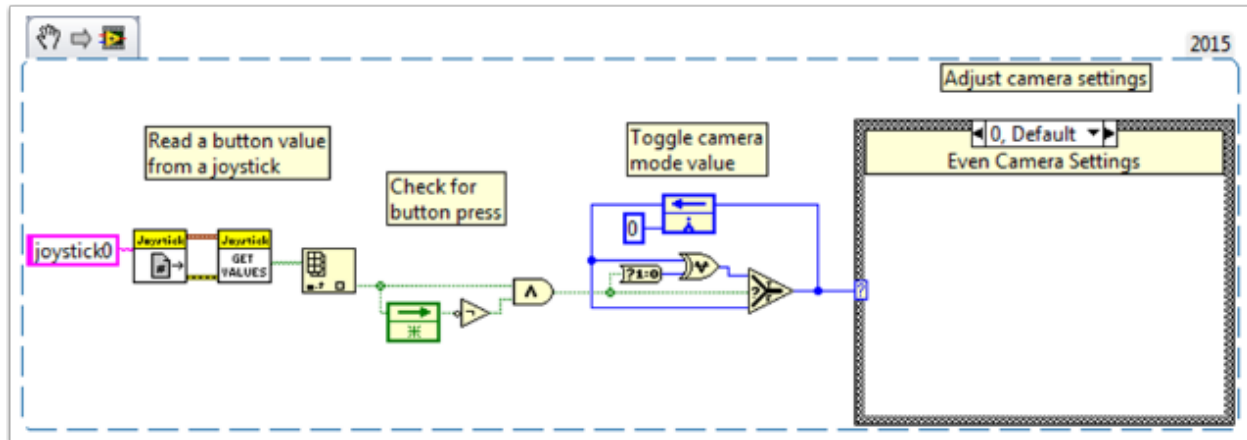
This code shows how to use a button to toggle between two distinct camera modes. The code consists of four stages.

In the first stage, the value of a button on the joystick is read.

Next, the current reading is compared to the previous reading using a **Feedback Node** and some Boolean arithmetic. Together, these ensure that the camera mode is only toggled when the button is initially pressed rather than toggling back and forth multiple times while the button is held down.

After that, the camera mode is toggled by masking the result of the second stage over the current camera mode value. This is called bit masking and by doing it with the **XOR** function the code will toggle the camera mode when the second stage returns true and do nothing otherwise.

Finally, you can insert the code for each camera mode in the case structure at the end. Each time the code is run, this section will run the code for the current camera mode.



3.2.5 LabVIEW Examples and Tutorials



Popular Tutorials

Autonomous Timed Movement Tutorial

- Move your robot autonomously based on different time intervals
- [See more on Autonomous Movement](#)

Basic Motor Control Tutorial

- Setup your roboRIO motor hardware and software
- Learn to setup the FRC Control System and FRC Robot Project
- [See more on Motor Control](#)

Image Processing Tutorial

- Learn basic Image Processing techniques and how to use NI Vision Assistant
- [See more on Cameras and Image Processing](#)

PID Control Tutorial

- What is PID Control and how can I implement it?

Command and Control Tutorial

- What is Command and Control?
- How do I implement it?

Driver Station Tutorial

- Get to know the FRC Driver Station

Test Mode Tutorial

- Learn to setup and use Test Mode

Looking for more examples and discussions? Search through more documents or post your own discussion, example code, or tutorial by [clicking here!](#) Don't forget to mark your posts with a tag!

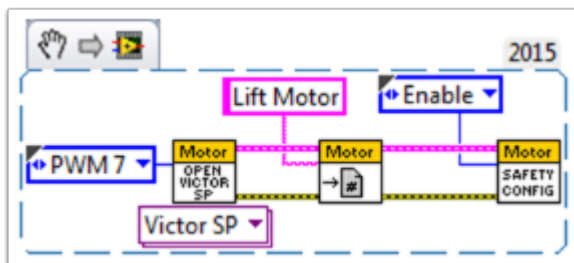
3.2.6 Add an Independent Motor to a Project



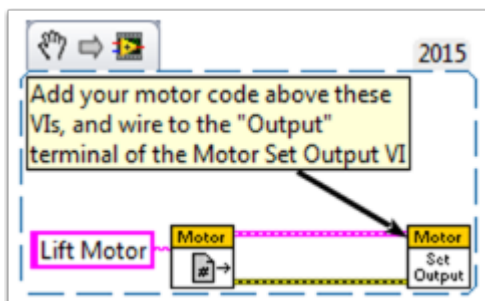
Once your drive that controls the wheels is all set, you might need to add an additional motor to control something completely independent of the wheels, such as an arm. Since this motor will not be part of your tank, arcade, or mecanum drive, you'll definitely want independent control of it.

These VI Snippets show how to set up a single motor in a project that may already contain a multi-motor drive. If you see the HAND>ARROW>LABVIEW symbol, just drag the image into your block diagram, and voila: code! Ok, here's how you do it.

FIRST, create a motor reference in the **Begin.vi**, using the **Motor Control Open VI** and **Motor Control Refnum Registry Set VI**. These can be found by right-clicking in the block diagram and going to **WPI Robotics Library>>RobotDrive>>Motor Control**. Choose your PWM line and name your motor. I named mine "Lift Motor" and connected it to PWM 7. (I also included and enabled the Motor Control Safety Config VI, which automatically turns off the motor if it loses connection.)

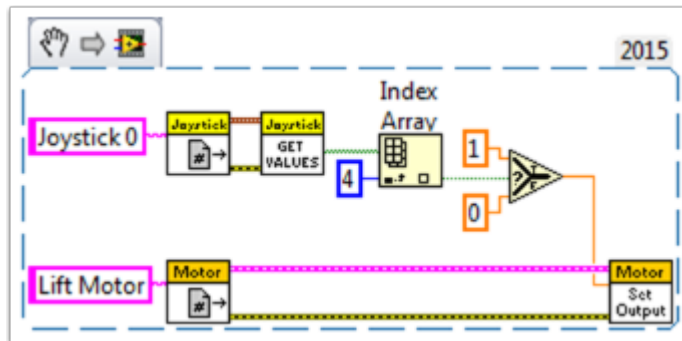


Now, reference your motor (the name has to be exact) in the **Teleop.vi** using the **Motor Control Refnum Registry Get VI** and tell it what to do with the **Motor Control Set Output VI**. These are in the same place as the above VIs.

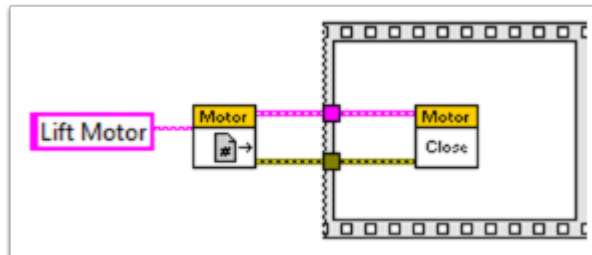


For example, the next snippet tells the Lift Motor to move forward if button 4 is pressed on Joystick 0 and to remain motionless otherwise. For me, button 4 is the left bumper on my

Xbox style controller (“Joystick 0”). For much more in-depth joystick button options, check out How to Use Joystick Buttons to Control Motors or Solenoids.



Finally, we need to close the references in the **Finish.vi** (just like we do with the drive and joystick), using the **Motor Control Refnum Registry Get VI** and **Motor Control Close VI**. While this picture shows the Close VI in a flat sequence structure by itself, we really want all of the Close VIs in the same frame. You can just put these two VIs below the other Get VIs and Close VIs (for the joystick and drive).



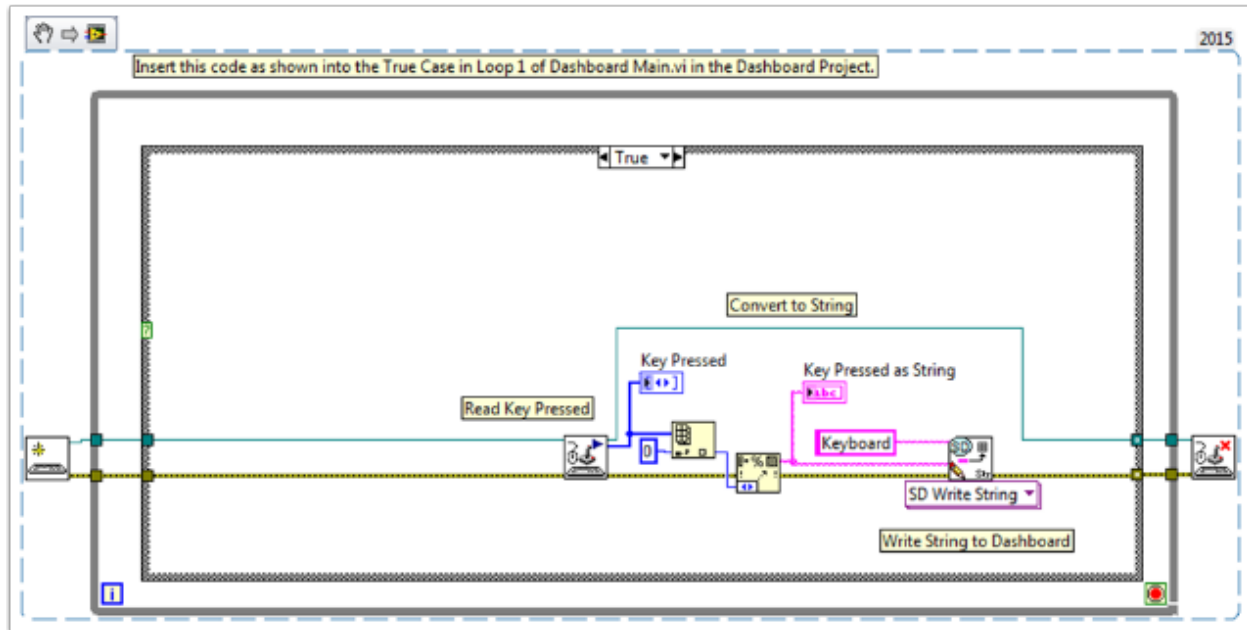
I hope this helps you program the best robot ever! Good luck!

3.2.7 Keyboard Navigation with the roboRIO



This example provides some suggestions for controlling the robot using keyboard navigation in place of a joystick or other controller. In this case, we use the A, W, S, and D keys to control two drive motors in a tank drive configuration.

The first VI Snippet is the code that will need to be included in the Dashboard Main VI. You can insert this code into the True case of Loop 1. The code opens a connection to the keyboard before the loop begins, and on each iteration it reads the pressed key. This information is converted to a string, which is then passed to the Teleop VI in the robot project. When Loop 1 stops running, the connection to the keyboard is closed.



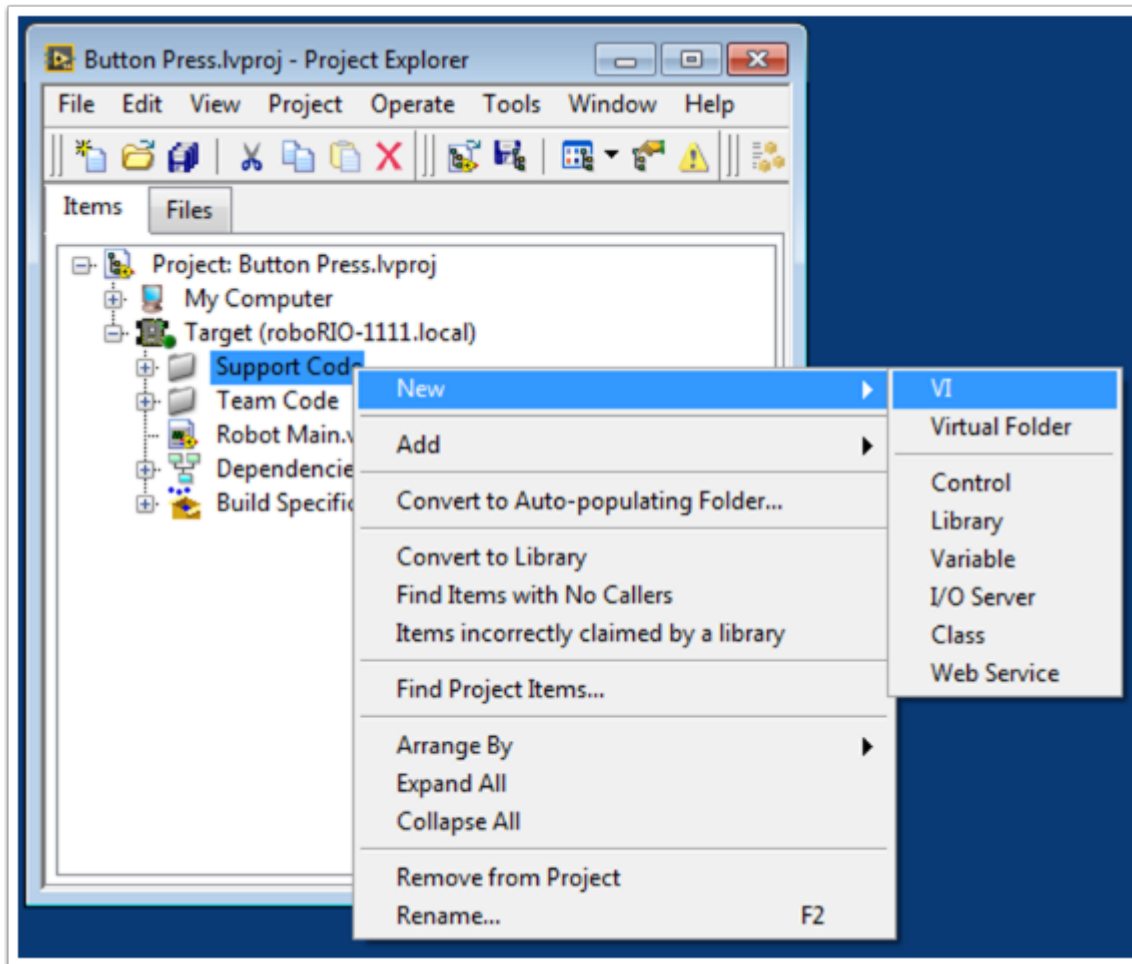
The second VI Snippet is code that should be included in the Teleop VI. This reads the string value from the Dashboard that indicates which key was pressed. A Case Structure then determines which values should be written to the left and right motors, depending on the key. In this case, W is forward, A is left, D is right, and S is reverse. Each case in this example runs the motors at half speed. You can keep this the same in your code, change the values, or add additional code to allow the driver to adjust the speed, so you can drive fast or slow as necessary. Once the motor values are selected, they are written to the drive motors, and motor values are published to the dashboard.

3.2.8 Making a One-Shot Button Press

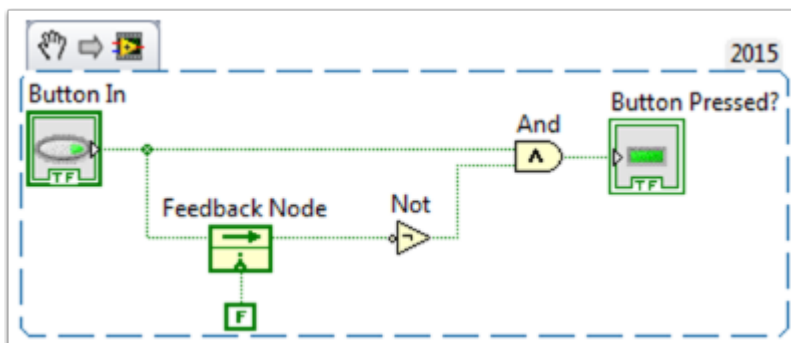


When using the Joystick Get Values function, pushing a joystick button will cause the button to read TRUE until the button is released. This means that you will most likely read multiple TRUE values for each press. What if you want to read only one TRUE value each time the button is pressed? This is often called a “One-Shot Button”. The following tutorial will show you how to create a subVI that you can drop into your Teleop.vi to do this.

First, create a new VI in the Support Code folder of your project.



Now on the block diagram of the new VI, drop in the following code snippet.

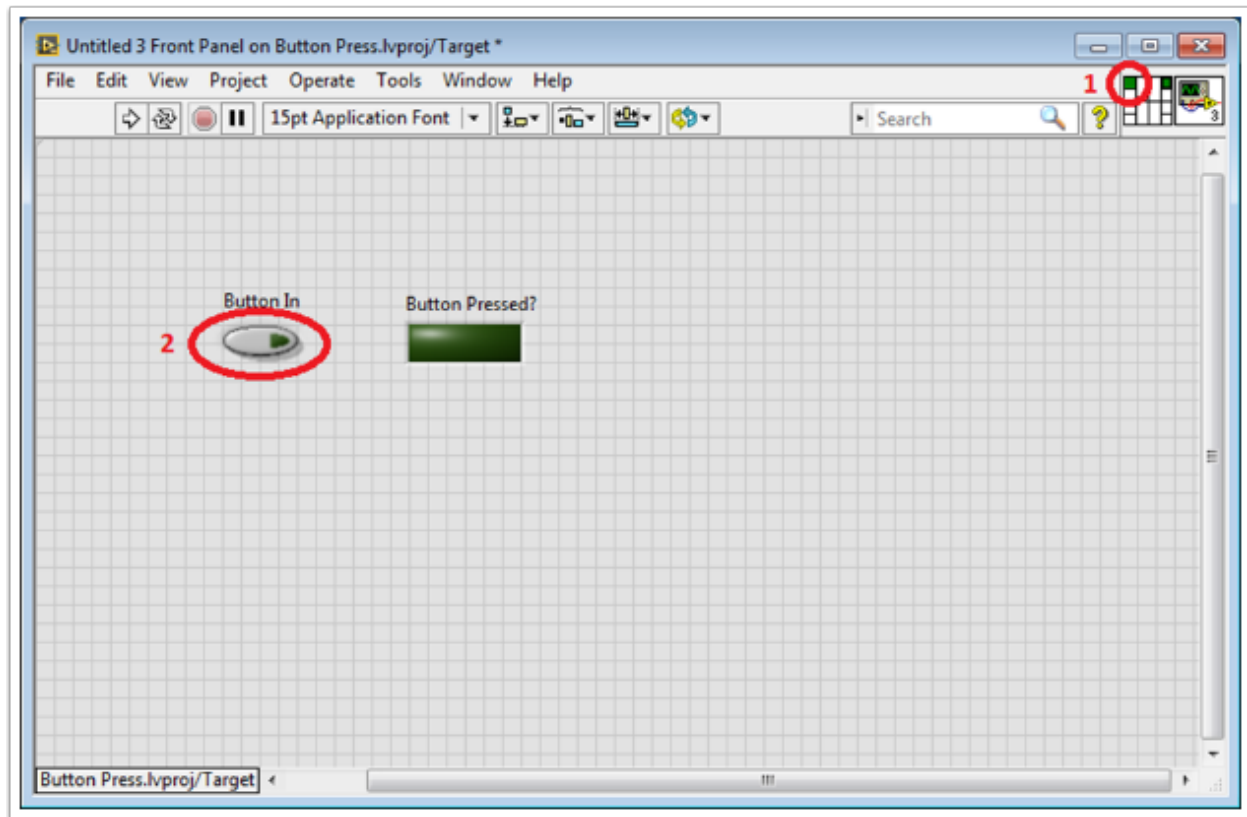


This code uses a function called the Feedback Node. We have wired the current value of the button into the left side of the feedback node. The wire coming out of the arrow of the feedback node represents the previous value of the button. If the arrow on your feedback node is going the opposite direction as shown here, right click to find the option to reverse the direction.

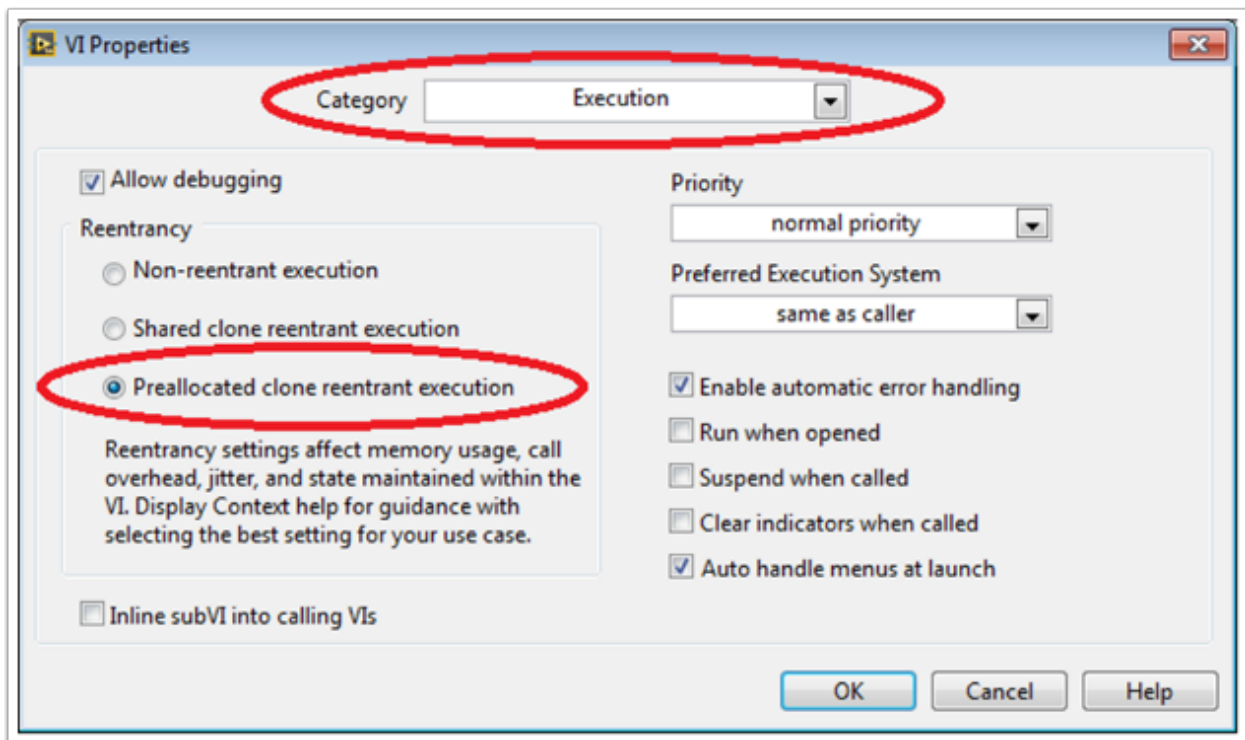
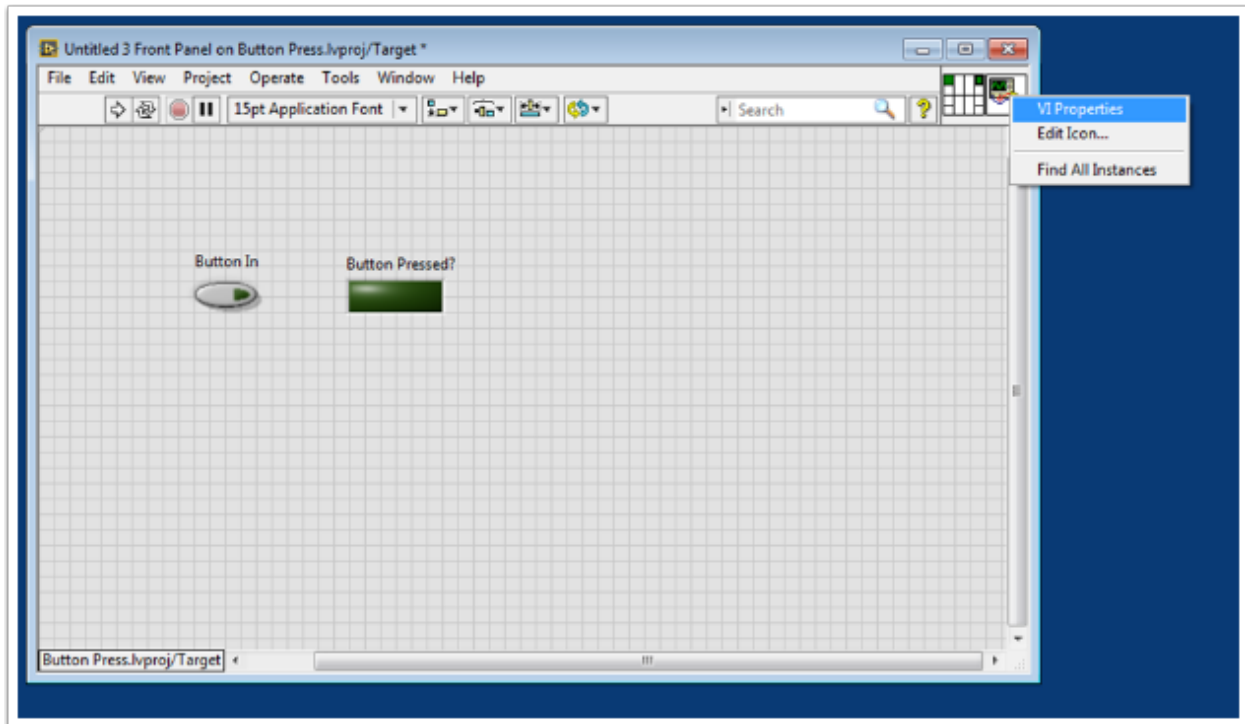
When a button is pressed, the value of the button goes from FALSE to TRUE. We want the output of this VI to be TRUE only when the current value of the button is TRUE, and the previous value of the button is FALSE.

Next we need to connect the boolean control and indicator to the inputs and outputs of the

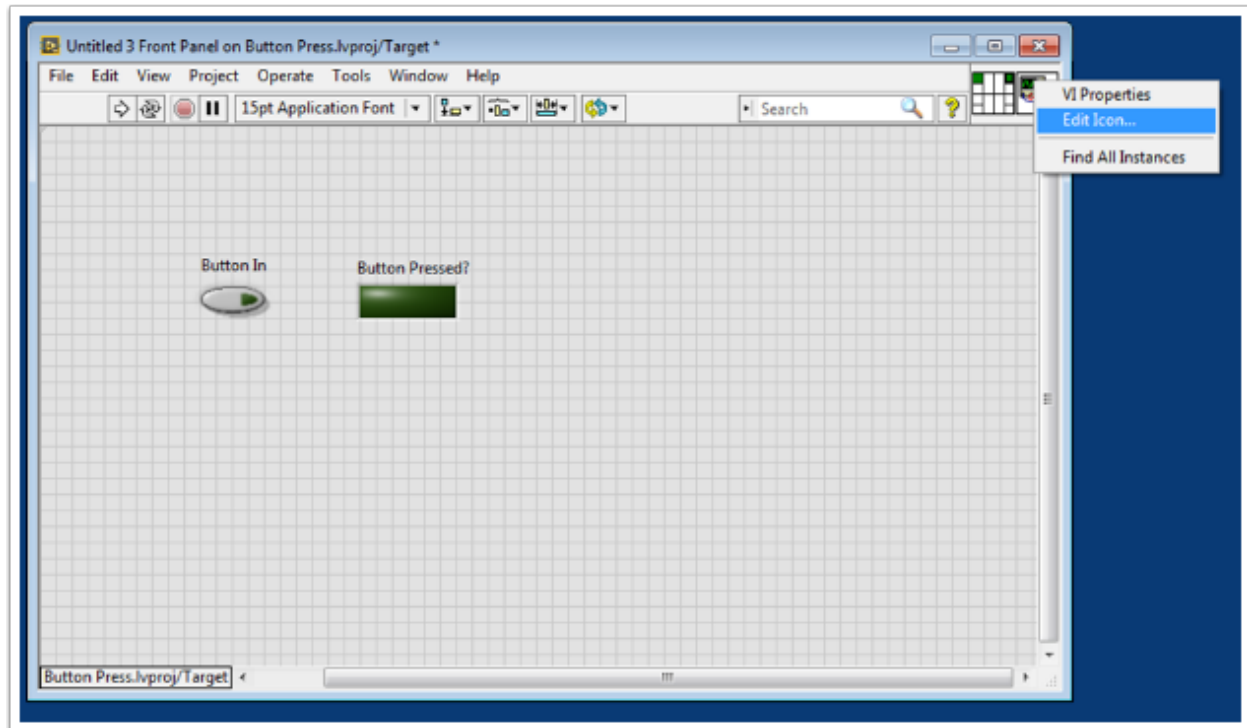
VI. To do this, first click the block on the connector pane, then click the button to connect the two (see the diagram below). Repeat this for the indicator.



Next, we need to change the properties of this VI so that we can use multiples of this VI in our TeleOp.vi. Right click the VI Icon and go to VI Properties. Then select the category "Execution" and select "Preallocated clone reentrant execution".

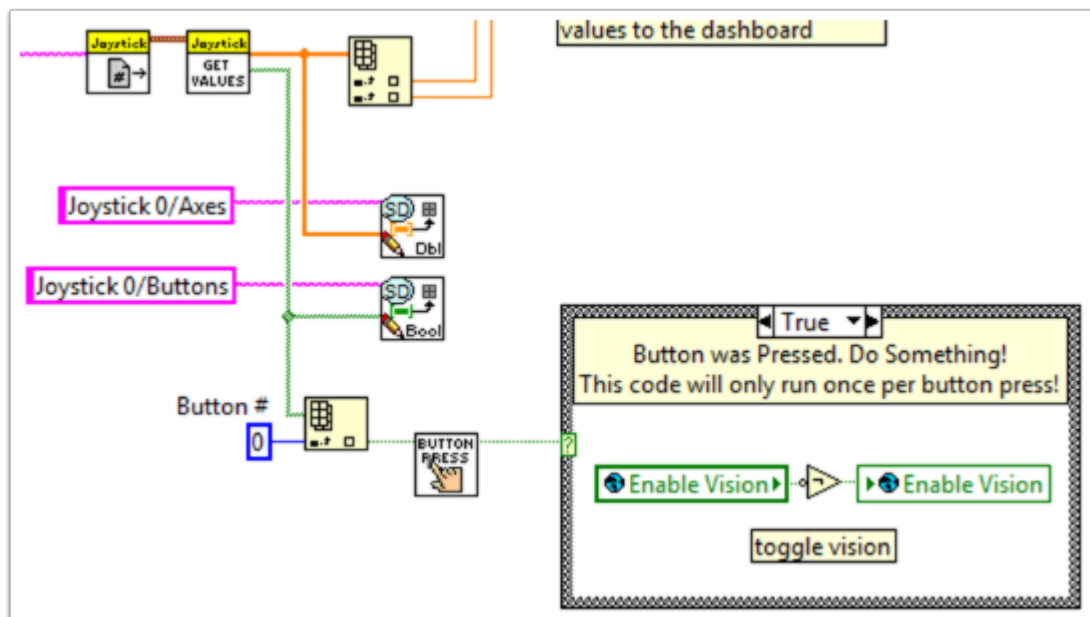


Lastly, we should change the VI Icon to be more descriptive of the VI's function. Right click the Icon and go to Edit Icon. Create a new Icon.



Finally, save the VI with a descriptive name. You can now drag and drop this VI from the Support Files folder into your TeleOp.vi. Here is a copy of the completed VI: Button_Press.vi

Here's an example of how you could use this VI.



3.2.9 Adding Safety Features to Your Robot Code

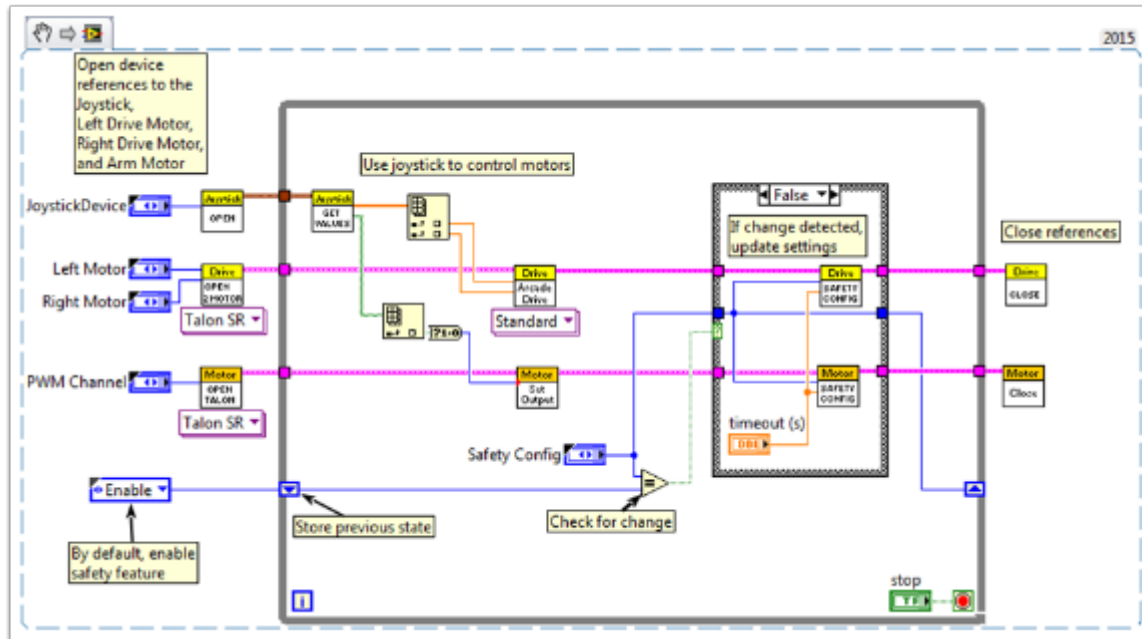


A common problem with complex projects is making sure that all of your code is executing when you expect it to. Problems can arise when tasks with high priority, long execution times, or frequent calls hog processing power on the roboRIO. This leads to what is known as “starvation” for the tasks that are not able to execute due to the processor being busy. In most cases this will simply slow the reaction time to your input from the joysticks and other devices. However, this can also cause the drive motors of your robot to stay on long after you try to stop them. To avoid any robotic catastrophes from this, you can implement safety features that check for task input starvation and automatically shut down potentially harmful operations.

There are built-in functions for the motors that allow easy implementation of safety checks. These functions are:

- Robot Drive Safety Configuration
- Motor Drive Safety Configuration
- Relay Safety Configuration
- PWM Safety Configuration
- Solenoid Safety Configuration
- Robot Drive Delay and Update Safety

In all of the Safety Configuration functions, you can enable and disable the safety checks while your programming is running and configure what timeout you think is appropriate. The functions keep a cache of all devices that have the safety enabled and will check if any of them have exceeded their time limit. If any has, all devices in the cache will be disabled and the robot will come to an immediate stop or have its relay/PWM/solenoid outputs turned off. The code below demonstrates how to use the Drive Safety Configuration functions to set a maximum time limit that the motors will receive no input before being shut off.



To test the safety shut-off, try adding a Wait function to the loop that is longer than your timeout!

The final function that relates to implementing safety checks-Robot Drive Delay and Update Safety-allows you to put the roboRIO in Autonomous Mode without exceeding the time limit. It will maintain the current motor output without making costly calls to the Drive Output functions, and will also make sure that the safety checks are regularly updated so that the motors will not suddenly stop.

Overall, it is highly recommended that some sort of safety check is implemented in your project to make sure that your robot is not unintentionally left in a dangerous state!

3.2.10 How to Use Joystick Buttons to Control Motors or Solenoids



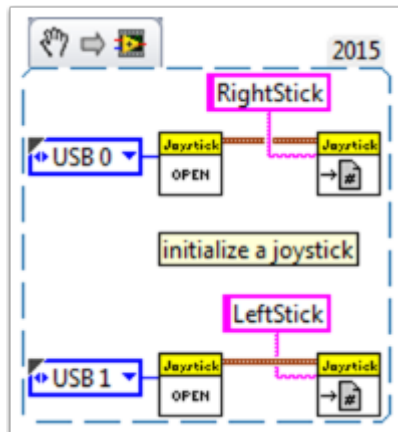
As we all get our drive systems working, we are moving on to connecting our auxiliary devices such as motors and solenoids. With this, we will generally use joystick buttons to control these devices. To get started with this, we'll go through several ways to control devices with joystick buttons.

Did you know that you can click and drag a VI Snippet from a document like this right into your LabVIEW code? Try it with the snippets in this document.

Setup:

No matter what the configuration, you'll need to add one, two, or more (if you're really excited) joysticks to the "Begin.vi". The first example uses 2 joysticks and the others only use one. Give each one a unique name so we can use it in other places, like the snippet below. I named

them “LeftStick” and “RightStick” because they are on the left and right sides of my desk. If your joysticks are already configured, great! You can skip this step.

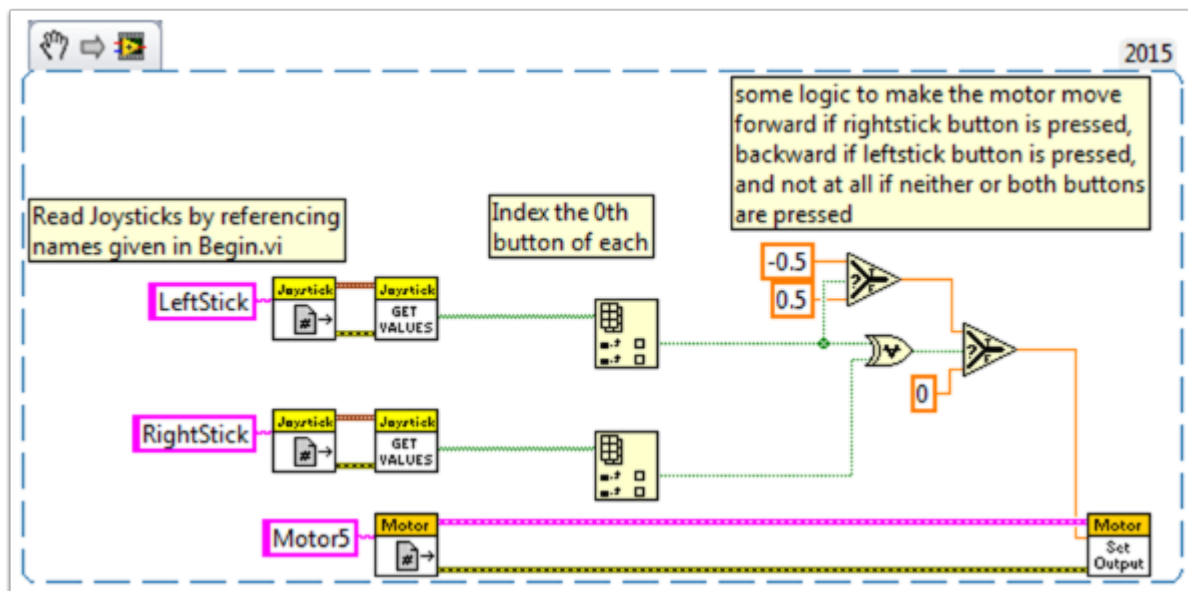


The rest of the code in this document will be placed in the “Teleop.VI” This is where we will be programming our joystick buttons to control different aspects of our motors or solenoids.

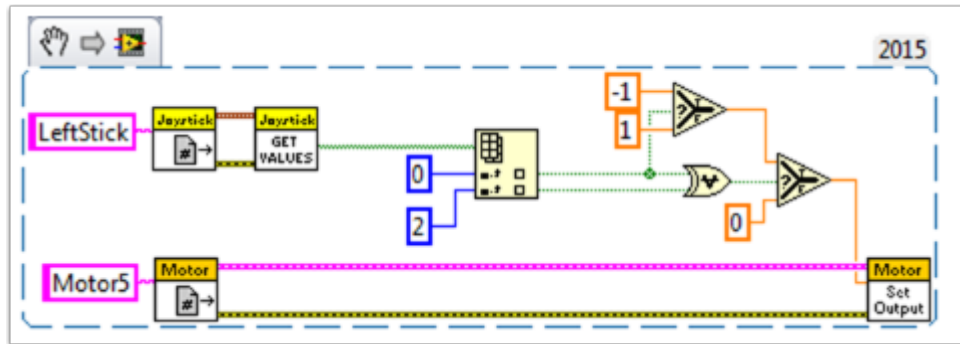
Scenario 1

“I want a motor to move one way when I press one button and the other way when I press a different button.”

This code uses button 0 on two different joysticks to control the same motor. If button 0 on LeftStick is pressed, the motor moves backward, and if button 0 on RightStick is pressed, the motor moves forward. If both buttons are pressed or neither button is pressed, the motor doesn't move. Here I named my motor reference “Motor5”, but you can name your motor whatever you want in the “Begin.vi”



You may want to use multiple buttons from the same joystick for control. For an example of this, look at the following VI snippet or the VI snippet in Scenario 2.



Here I used joystick buttons 0 and 2, but feel free to use whatever buttons you need.

Scenario 2

“I want different joystick buttons move at various speeds.”

This example could be helpful if you need to have one motor do different things based on the buttons you press. For instance, let’s say my joystick has a trigger (button 0) and 4 buttons on top (buttons 1 through 4). In this case, the following buttons should have the following functions:

- button 1 - move backward at half speed
- button 2 - move forward at half speed
- button 3 - move backward at 1/4 speed
- button 4 - move forward at 1/4 speed
- trigger - full speed ahead! (forward at full speed)

We would then take the boolean array from the “JoystickGetValues.vi” and wire it to a “Boolean Array to Number” node (Numeric Palette-Conversion Palette). This converts the boolean array to a number that we can use. Wire this numeric to a case structure.

Each case corresponds to a binary representation of the values in the array. In this example, each case corresponds to a one-button combination. We added six cases: 0 (all buttons off), 1 (button 0 on), 2 (button 1 on), 4 (button 2 on), 8 (button 3 on), and 16 (button 4 on). Notice we skipped value 3. 3 would correspond to buttons 0 and 1 pressed at the same time. We did not define this in our requirements so we’ll let the default case handle it.

It might be helpful to review the LabVIEW 2014 Case Structure Help document here:

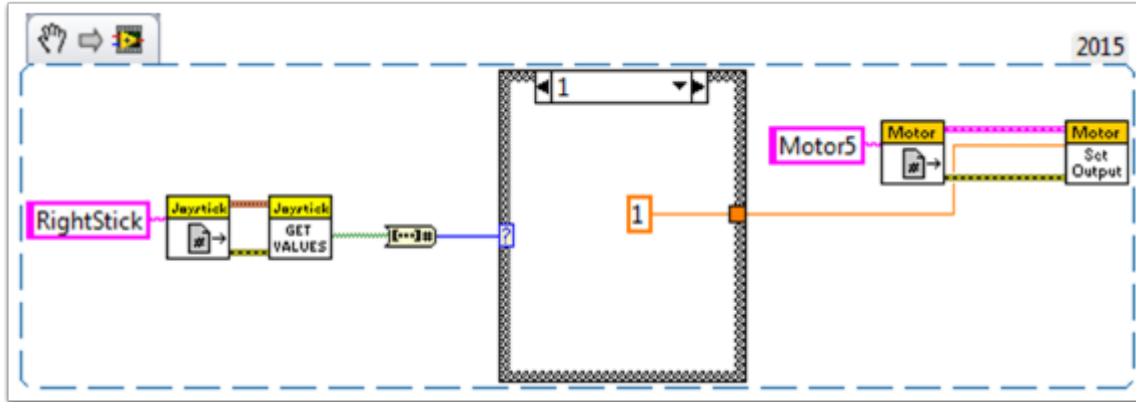
https://zone.ni.com/reference/en-XX/help/371361L-01/glang/case_structure/

There are also 3 Community Tutorials on case structures here:

<https://forums.ni.com/t5/Curriculum-and-Labs-for/Unit-3-Case-Structures-Lesson-1/ta-p/3505945?profile.language=en>

<https://forums.ni.com/t5/Curriculum-and-Labs-for/Unit-3-Case-Structures-Lesson-2/ta-p/3505933?profile.language=en>

<https://forums.ni.com/t5/Curriculum-and-Labs-for/Unit-3-Case-Structures-Lesson-3/ta-p/3505979?profile.language=en>

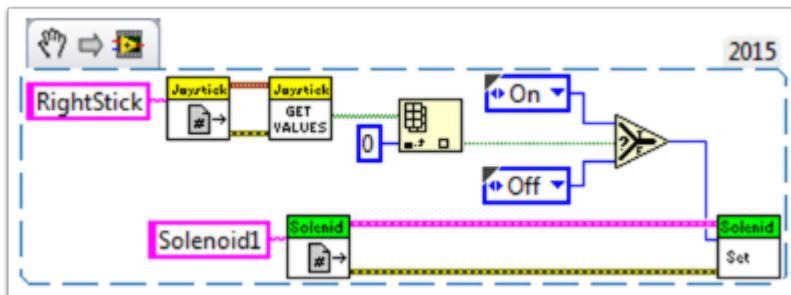


Since our requirements were simple, we only need a single constant in each case. For case 1 (full ahead) we use a 1, for case 2 (half back) we use a -0.5, etc. We can use any constant value between 1 and -1. I left case 0 as the default so if multiple buttons are pressed (any undefined state was reached) the motor will stop. You of course are free to customize these states however you want.

Scenario 3

“I want to control a solenoid with my joystick buttons.”

By now, we are familiar with how the joystick outputs the buttons in an array of booleans. We need to index this array to get the button we are interested in, and wire this boolean to a select node. Since the “Solenoid Set.vi” requires a Enum as an input, the easiest way to get the enum is to right click the “Value” input of the “Solenoid Set.vi” and select “Create Constant”. Duplicate this constant and wire one copy to the True terminal and one to the False terminal of the select node. Then wire the output of the select node to the “Value” input of the solenoid VI.



Happy Roboting!

3.2.11 Local and Global Variables in LabVIEW for FRC



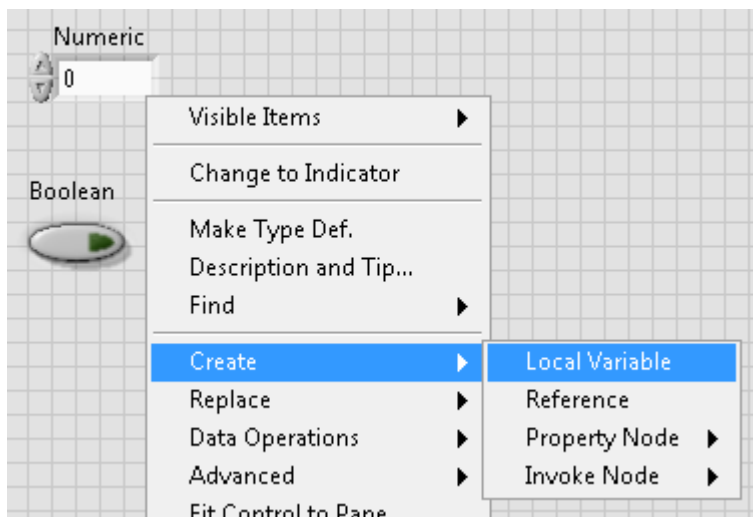
This example serves as an introduction to local and global variables, how they are used in the default LabVIEW for FRC Robot Project, and how you might want to use them in your project.

Local variables and global variables may be used to transfer data between locations within the same VI (local variables) or within different VI's (global variables), breaking the conventional [Data Flow Paradigm](#) for which LabVIEW is famous. Thus, they may be useful when, for whatever reason, you cannot wire the value directly to the node to another.

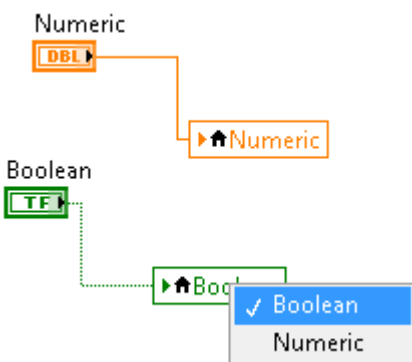
Note: One possible reason may be that you need to pass data between consecutive loop iterations; Miro_T covered this [in this post](#). It should also be noted that the [feedback node](#) in LabVIEW may be used as an equivalent to the shift register, although that may be a topic for another day!

Introduction to Local and Global Variables

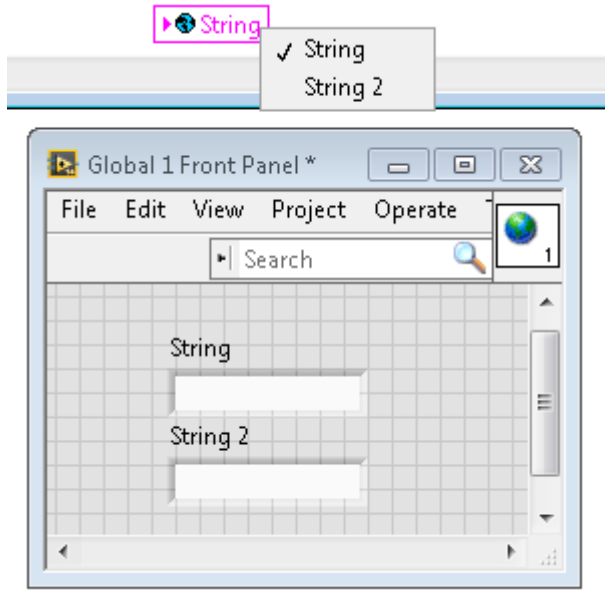
Local variables may be used within the same VI. Create a local variable by right-clicking a control or indicator on your Front Panel:



You may create a local variable from the Structures palette on the block diagram as well. When you have multiple local variables in one VI, you can left-click to choose which variable it is:



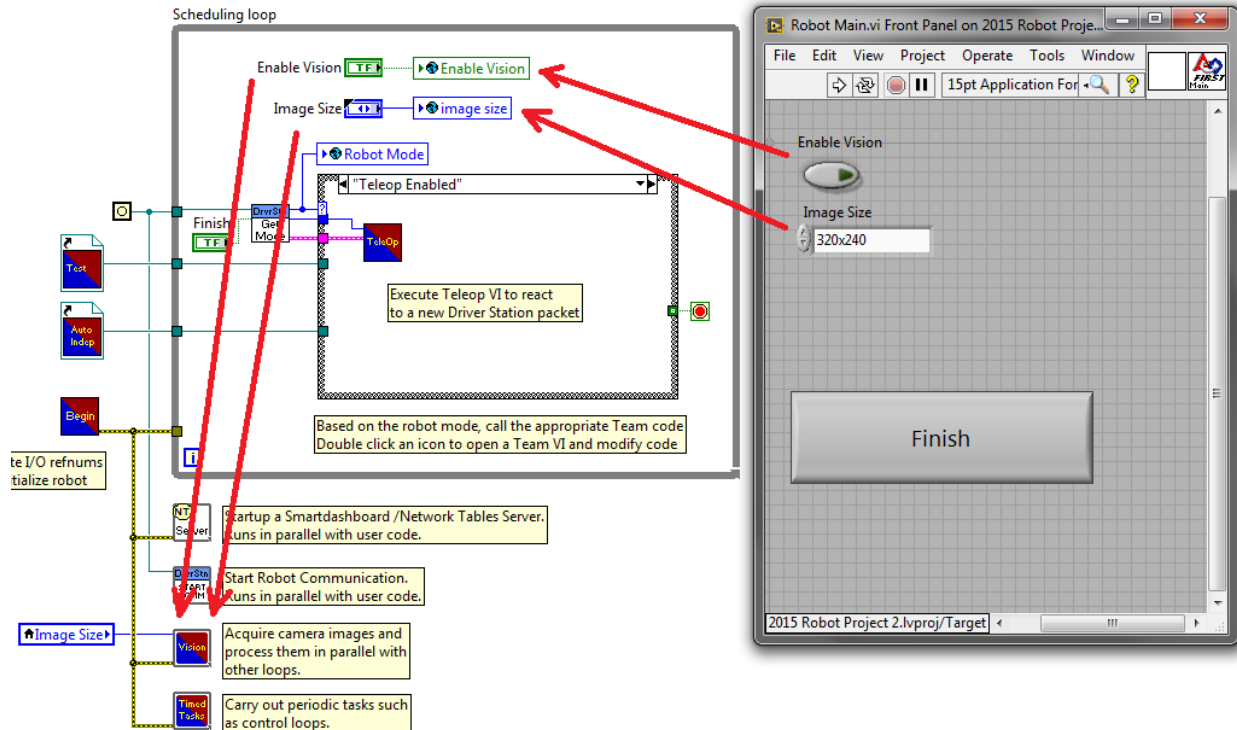
Global variables are created slightly differently. Add one to the block diagram from the Structures palette, and notice that when you double-click it, it opens a separate front panel. This front panel does not have a block diagram, but you add as many entities to the front panel as you wish and save it as a *.vi file:



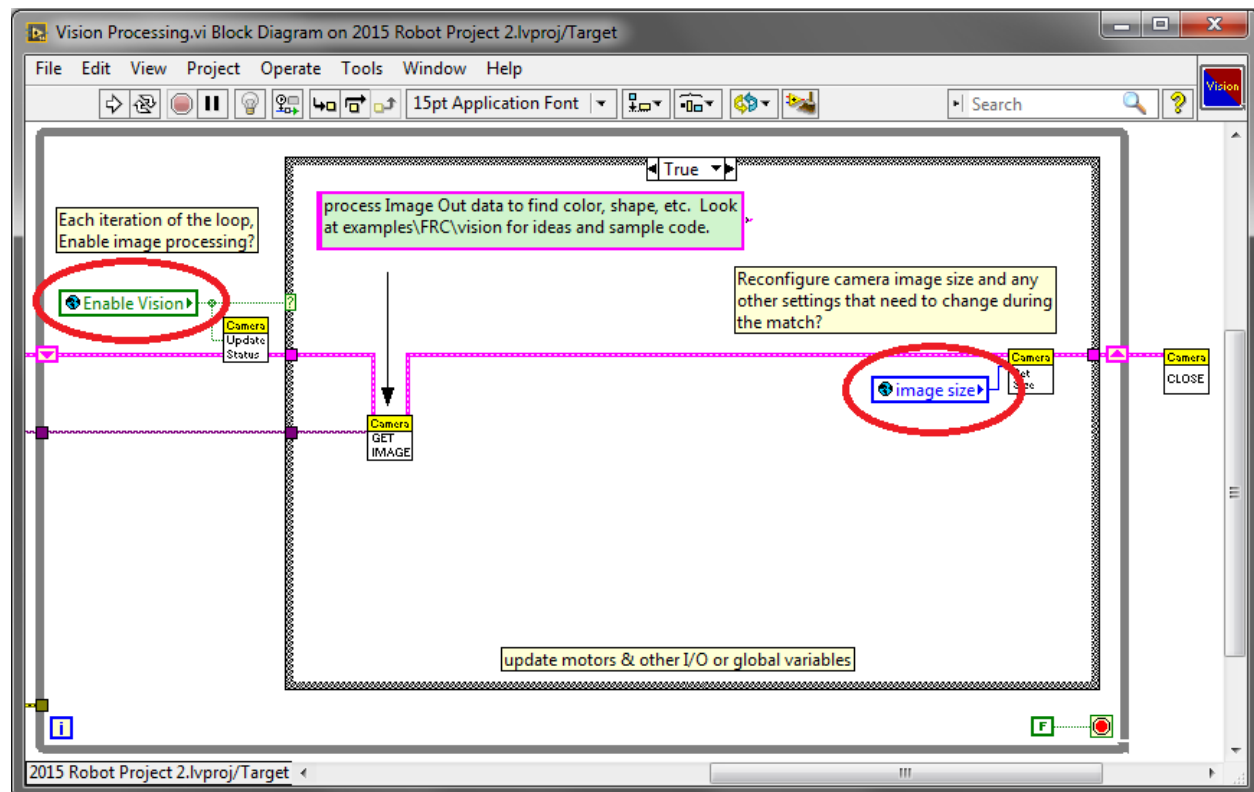
Note: Be very careful to avoid race conditions when using local and global variables! Essentially, make sure that you are not accidentally writing to the same variable in multiple locations without a way to know to which location it was last written. For a more thorough explanation, see [this help document](#)

How They are Used in the Default LabVIEW for FRC Robot Project

Global variables for “Enable Vision” and “Image Size” are written to during each iteration of the Robot Main VI...



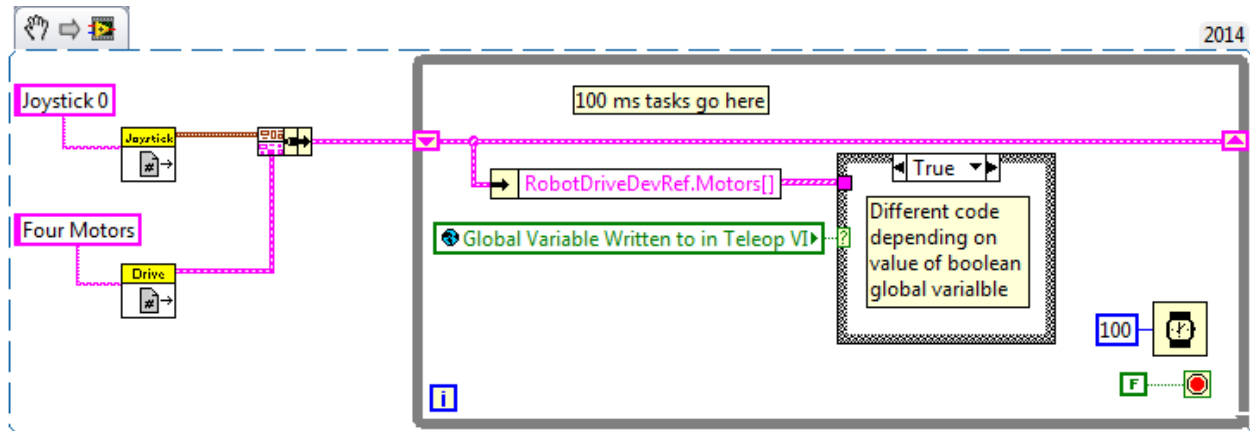
... And then read in each iteration of the Vision Processing VI:



This allows the user, when deploying to Robot Main VI from the LabVIEW Development Environment, to enable/disable vision and change the image size from Robot Main's Front Panel.

How Can You Use Them in Your Project?

Check out the block diagram for the Periodic Tasks VI. Perhaps there is some value, such as a boolean, that may be written to a global variable in the Teleop VI, and then read from in the Periodic Tasks VI. You can then decide what code or values to use in the Periodic Tasks VI, depending on the boolean global variable:



3.2.12 Using the Compressor in LabVIEW



This snippet shows how to set up your roboRIO project to use the Pneumatic Control Module (PCM). The PCM automatically starts and stops the compressor when specific pressures are measured in the tank. In your roboRIO program, you will need to add the following VIs.

For more information, check out the following links:

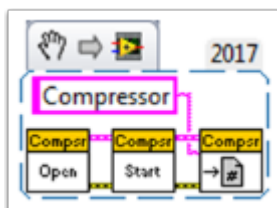
FRC Pneumatics Manual

PCM User's Guide

Pneumatics Step by Step for the roboRIO

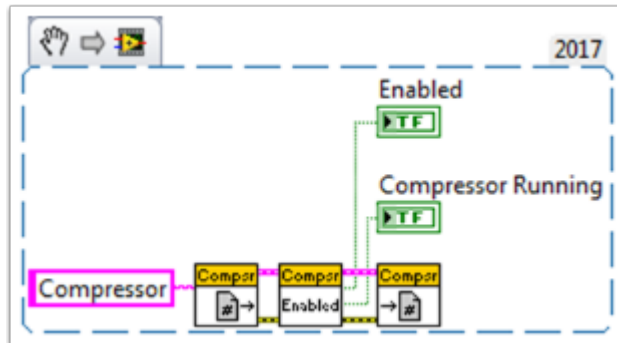
Begin VI

Place this snippet in the `Begin.vi`.



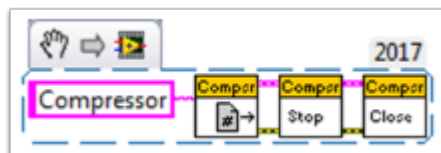
Teleop VI

Place this snippet in the Teleop.vi. This portion is only required if you are using the outputs for other processes.



Finish VI

Place this snippet in Close Refs, save data, etc. frame of the Finish.vi.



4.1 Visão geral dos atuadores

Essa sessão discute sobre o controle de motor e pneumático através de rápidos controladores, solenóides e pneumático, e sua interface com C++ e Java WPILib.

4.1.1 Controladores de velocidade

Um controlador de velocidade é responsável em seu robô para fazer os motores mover. Para brushed DC motors como também para CIMs ou 775s, o controlador de velocidade regula a voltagem do que o motor recebe, muito parecido com uma lâmpada. Para o controlador de velocidade Brushless como também para o Spark MAX, os controladores regulam a energia entregue para cada “fase” do motor.

Conectando um controlador de motor BRUSHLESS direto a energia, como também para a convencional controlador de motor brushed, irá destruir o motor!

Controladores de motores permitidos na FRC

Controladores de velocidade vêm em muitas formas, tamanhos e conjuntos de recursos. Isso é a lista completa de Controladores de Velocidade permitidos na FRC como de Janeiro 2020:

- DMC 60/DMC 60c Motor Controller (P/N: 410-334-1, 410-334-2)
- Jaguar Motor Controller (P/N: MDL-BDC, MDL-BDC24, and 217-3367) connected to PWM only
- Nidec Dynamo BLDC Motor with Controller to control integral actuator only (P/N 840205-000, am-3740)
- SD540 Motor Controller (P/N: SD540x1, SD540x2, SD540x4, SD540Bx1, SD540Bx2, SD540Bx4, SD540C)
- Spark Motor Controller (P/N: REV-11-1200)
- Spark MAX Motor Controller (P/N: REV-11-2158)

- Talon FX Motor Controller (P/N: 217-6515, 19-708850, am-6515, am-6515_Short) for controlling integral Falcon 500 only
- Talon Motor Controller (P/N: CTRE_Talon, CTRE_Talon_SR, and am-2195)
- Talon SRX Motor Controller (P/N: 217-8080, am-2854, 14-838288)
- Victor 884 Motor Controller (P/N: VICTOR-884-12/12)
- Victor 888 Motor Controller (P/N: 217-2769)
- Victor SP Motor Controller (P/N: 217-9090, am-2855, 14-868380)
- Victor SPX Motor Controller (P/N: 217-9191, 17-868388, am-3748)
- Venom Motor with Controller (P/N BDC-10001) for controlling integral motor only

4.1.2 Pneumática

Pneumática é um caminho rápido e fácil para fazer alguma coisa que é em um estado ou outro usando compressão de ar. Para informações em operando pneumática, veja [Operando cilindros pneumáticos](#).

Relay Modules permitidos na FRC

- Spike H-Bridge Relay (P/N: 217-0220 and SPIKE-RELAY-H)
- Automation Direct Relay (P/N: AD-SSR6M12-DC200D, AD-SSR6M25-DC200D, AD-SSR6M40-DC200D)

Controladores pneumáticos permitidos na FRC

- Pneumatics Control Module (P/N: am-2858, 217-4243)

4.2 Operando cilindros pneumáticos

4.2.1 Usando o sistema de controle da FRC para controlar a pneumática

Note: Pneumatics Control Module (PCM) é a um dispositivo que utiliza a comunicação CAN que tem controle total do compressor e até 8 módulos solenóides . A PCM é integrada em WPILib através de uma série de classes que simplificam o uso. O controle de circuito fechado do compressor e pressão é tratado pelo hardware da PCM e dos solenóides são manipulados pela Solenóide atualizada, que agora, controla os canais do solenóide na PCM. Além disso, um módulo PCM pode ser usado onde os módulo correspondentes aos solenóides são diferenciados pelo número do módulo nos construtores das classes de Solenóide e Compressor.



O Módulo de Controle Pneumático da CTR Electronics é responsável por regular a pressão do robô usando um switch de pressão e um compressor e ligando e desligando os solenóides. A PCM se comunica com o roboRIO através do CAN. Para obter mais informações, consulte a Visão geral do hardware do sistema de controle FRC.

4.2.2 Números de módulos PCM

Os módulos da PCM são identificados pelo seus Node ID. O padrão Node ID para PCMs é 0. Se estiver usando um único PCM na linha CAN é recomendado deixá-lo no padrão Node ID.

4.2.3 Gerando e armazenando pressão

Na FRC, a pressão é criada usando um compressor pneumático e armazenada em tanques pneumáticos. O compressor não precisa necessariamente estar no robô, mas deve ser alimentado pela PCM do robô. O modo de “Closed Loop” no compressor é ativado por padrão, e é ele não recomenda que as equipes de alterar essa configuração. Quando o “closed loop control” está ativado, o PCM liga automaticamente o compressor quando o switch pressão está fechado (abaixo do limiar de pressão) e o desliga quando o switch pressão está aberto (~120PSI). Quando o “loop control” está desativado, o compressor não liga. Usando um compressor, os usuários podem consultar o status do compressor. O estado (atualmente ativado ou desativado), o estado do switch pressão e a corrente do compressor podem ser consultados no objeto Compressor.

Note: A PCM da Cross the Road Electronics permite para integrar closed loop control do compressor. Criando qualquer instância de um objeto Solenóide ou Solenóide Duplo habilitará

o controle Compressor no PCM correspondente. O objeto Compressor é necessário apenas se você desejar desativar o compressor ou consultar o status do compressor.

4.2.4 Controle Solenóide

As equipes da FRC usam solenóides para realizar várias tarefas, desde a troca de caixas de velocidades até a operação de mecanismos de robô. Um solenóide é uma válvula usada para ativar eletronicamente uma linha de ar pressurizada “ligada” ou “desligada”. Para obter mais informações sobre solenóides, consulte este artigo da Wikipedia <https://en.wikipedia.org/wiki/Solenoid_valve>’. Os solenóides são controlados pelo módulo de controle pneumático do robô, ou PCM, que por sua vez é conectado ao roboRIO do robô via CAN. A maneira mais fácil de ver o estado de um solenóide é através do pequeno LED vermelho (que indica se a válvula está “ligada” ou não), e os solenóides podem ser acionados manualmente quando não forem energizados com o pequeno botão adjacente ao LED.

Os solenóides de ação simples aplicam ou liberam a pressão de uma única porta de saída. Eles geralmente são usados quando uma força externa fornece a ação de retorno do cilindro (mola, gravidade, mecanismo separado) ou em pares para atuar como um solenóide duplo. Um solenóide duplo alterna o fluxo de ar entre duas portas de saída (muitas também têm uma posição central em que nenhuma saída é ventilada ou conectada à entrada). As válvulas solenóides duplas são comumente usadas quando você deseja controlar as ações de extensão e retração de um cilindro usando a pressão do ar. As válvulas solenóides duplas têm duas entradas elétricas que se conectam novamente a dois canais separados na ruptura do solenóide.

Os módulos PCM são identificados pelo seu ID de dispositivo CAN. O ID CAN padrão para PCMs é 0. Se você estiver usando um único PCM no barramento, é recomendável deixá-lo no ID CAN padrão. Esse ID pode ser alterado com o aplicativo Phoenix Tuner, além de outras informações de depuração. O Phoenix Tuner pode ser baixado no GitHub.<<https://github.com/CrossTheRoadElec/Phoenix-Releases>>’. Para obter mais informações sobre como definir IDs CAN do PCM, consulte Atualização e configuração do módulo de controle pneumático e do painel de distribuição de energia.

4.2.5 Solenóides únicos no WPILib

Solenóides únicos no WPILib são controlados usando a classe Solenóide. Para construir um objeto Solenóide, simplesmente passe o número da porta desejada (assume CAN ID 0) ou CAN ID e número da porta ao construtor. Para definir o valor do conjunto de chamadas do solenóide (true) para ativar ou definir (false) para desativar a saída do solenóide.

4.2.6 Solenóides duplos no WPILib

Os solenóides duplos são controlados pela classe Solenóides duplos no WPILib. Eles são construídos de maneira semelhante ao solenóide único, mas agora existem dois números de porta a serem passados ao construtor, um canal direto (primeiro) e um canal reverso (segundo). O estado da válvula pode então ser definido como kOff (nenhuma saída ativada), kForward (canal direto ativado) ou kReverse (canal reverso ativado). Além disso, o PCM CAN ID pode ser passado para o Solenóide duplo se as equipes tiverem um PCM CAN ID não padrão.

4.3 Usando controladores de motor no código

Os controladores de motores têm dois tipos principais: CAN e PWM. Um controlador CAN pode enviar informações de status mais detalhadas de volta ao roboRIO, enquanto um controlador PWM pode ser definido apenas como um valor. Para obter informações sobre o uso desses motores com as classes de transmissão WPI, consulte [Usando WPILib para conduzir seu robô](#).

4.3.1 Usando controladores de velocidade PWM

Os controladores de velocidade PWM podem ser controlados da mesma forma que um controlador de velocidade CAN. Para obter um plano de fundo mais detalhado sobre *how* eles funcionam, consulte [Controladores de velocidade PWM em profundidade](#). Para usar um controlador de velocidade PWM, basta usar a classe de controlador de velocidade apropriada fornecida pela WPI e fornecer a porta na qual os controladores de velocidade estão conectados no roboRIO. Todos os controladores de motor aprovados têm classes WPI fornecidas para eles.

4.3.2 Controladores CAN motor

Um punhado de controladores de velocidade CAN está disponível em fornecedores como CTR Electronics e REV Robotics.

SPARK MAX

Para obter informações sobre o SparkMAX CAN Speed Controller, que pode ser usado no modo CAN ou PWM, consulte os [software resources](#) e [example code](#).

Controladores de motor CTRE CAN

Consulte a documentação da CTR de terceiros no software Phoenix para obter informações mais detalhadas. A documentação está disponível [here](#).

4.4 Controladores de velocidade PWM em profundidade

O WPILib possui amplo suporte para o controle do motor. Existem várias classes que representam diferentes tipos de controladores de velocidade e servos. Atualmente, existem duas classes de controladores de velocidade, controladores de motor baseados em PWM e controladores de motor baseados em CAN. O WPILib também contém classes compostas (como DifferentialDrive) que permitem controlar vários motores com um único objeto. Este artigo abordará os detalhes dos controladores de motor PWM; Controladores CAN e classes compostas serão abordados em artigos separados.

4.4.1 Controladores PWM, breve teoria de operação

PWM significa Modulação por largura de pulso. Para controladores de motor, o PWM pode se referir ao sinal de entrada e ao método que o controlador usa para controlar a velocidade do motor. Para controlar a velocidade do motor, o controlador deve variar a tensão de entrada percebida do motor. Para fazer isso, o controlador liga e desliga a tensão total de entrada muito rapidamente, variando a quantidade de tempo em que se baseia com base no sinal de controle. Devido às constantes de tempo mecânicas e elétricas dos tipos de motores usados no FRC, essa comutação rápida produz um efeito equivalente ao da aplicação de uma tensão mais baixa fixa (50% de comutação produz o mesmo efeito que a aplicação de ~ 6V).

O sinal PWM que os controladores usam para uma entrada é um pouco diferente. Mesmo nos limites da faixa do sinal (avanço máximo ou reverso máximo), o sinal nunca se aproxima de um ciclo de trabalho de 0% ou 100%. Em vez disso, os controladores usam um sinal com um período de 5 ms ou 10 ms e uma largura de pulso no ponto médio de 1,5 ms. Muitos dos controladores usam o tempo típico do controlador RC de 1ms a 2ms.

4.4.2 Valores de saída brutos versus dimensionados

Em geral, todas as classes de controladores de motor no WPILib assumem um valor escalonado de -1,0 a 1,0 como saída para um atuador. O módulo PWM no FPGA no roboRIO é capaz de gerar sinais PWM com períodos de 5, 10 ou 20ms e pode variar a largura do pulso em 2000 etapas de ~ 0,001ms cada em torno do ponto médio (1000 etapas em cada direção em torno do ponto médio)) Os valores brutos enviados para este módulo estão nesta faixa de 0-2000, sendo 0 um caso especial que mantém o sinal baixo (desativado). A classe para cada controlador de motor contém informações sobre quais são os valores típicos de limite (min, max e cada lado da faixa morta), bem como o ponto médio típico. O WPILib pode então usar esses valores para mapear o valor escalado na faixa adequada para o controlador do motor. Isso permite que o código alterne perfeitamente entre diferentes tipos de controladores e abstraia os detalhes da sinalização específica. Controladores de velocidade de calibração

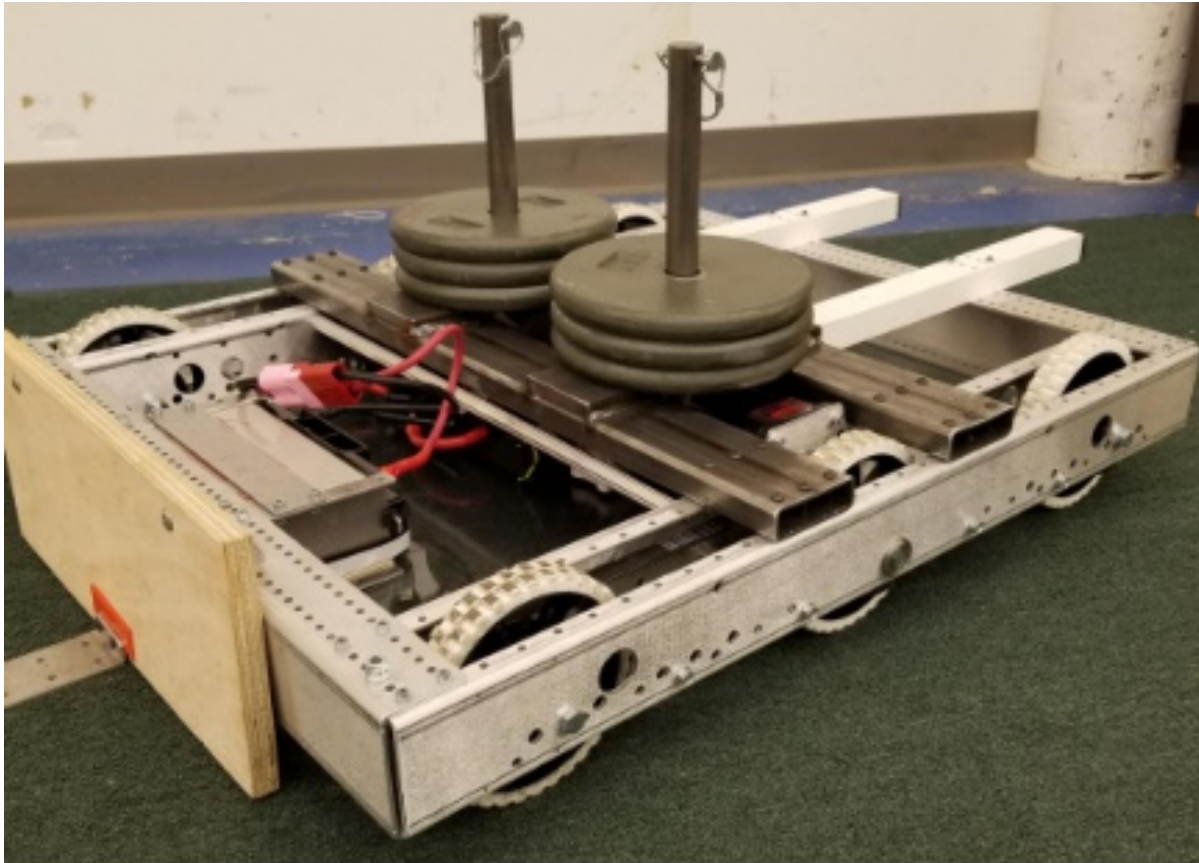
Portanto, se o WPILib lida com toda essa escala, por que você precisaria calibrar seu controlador de velocidade? Os valores que o WPILib usa para dimensionar são aproximados com base na medição de várias amostras de cada tipo de controlador. Devido a uma variedade de fatores, o tempo de um controlador de velocidade individual pode variar um pouco. Para eliminar definitivamente o “zumbido” (sinal do ponto médio interpretado como um leve movimento em uma direção) e conduzir o controlador até o extremo, é recomendável calibrar os controladores. Em geral, o procedimento de calibração para cada controlador envolve colocar o controlador no modo de calibração e, em seguida, direcionar o sinal de entrada para cada extremo e, em seguida, retornar ao ponto médio. Para obter exemplos de como usar esses controladores de velocidade no seu código, consulte [Using Motor Controllers in Code/Using PWM Speed Controllers](#)

4.5 Usando WPILib para conduzir seu robô

O WPILib contém muitas explicações para ajudar seu robô a dirigir mais rápido.

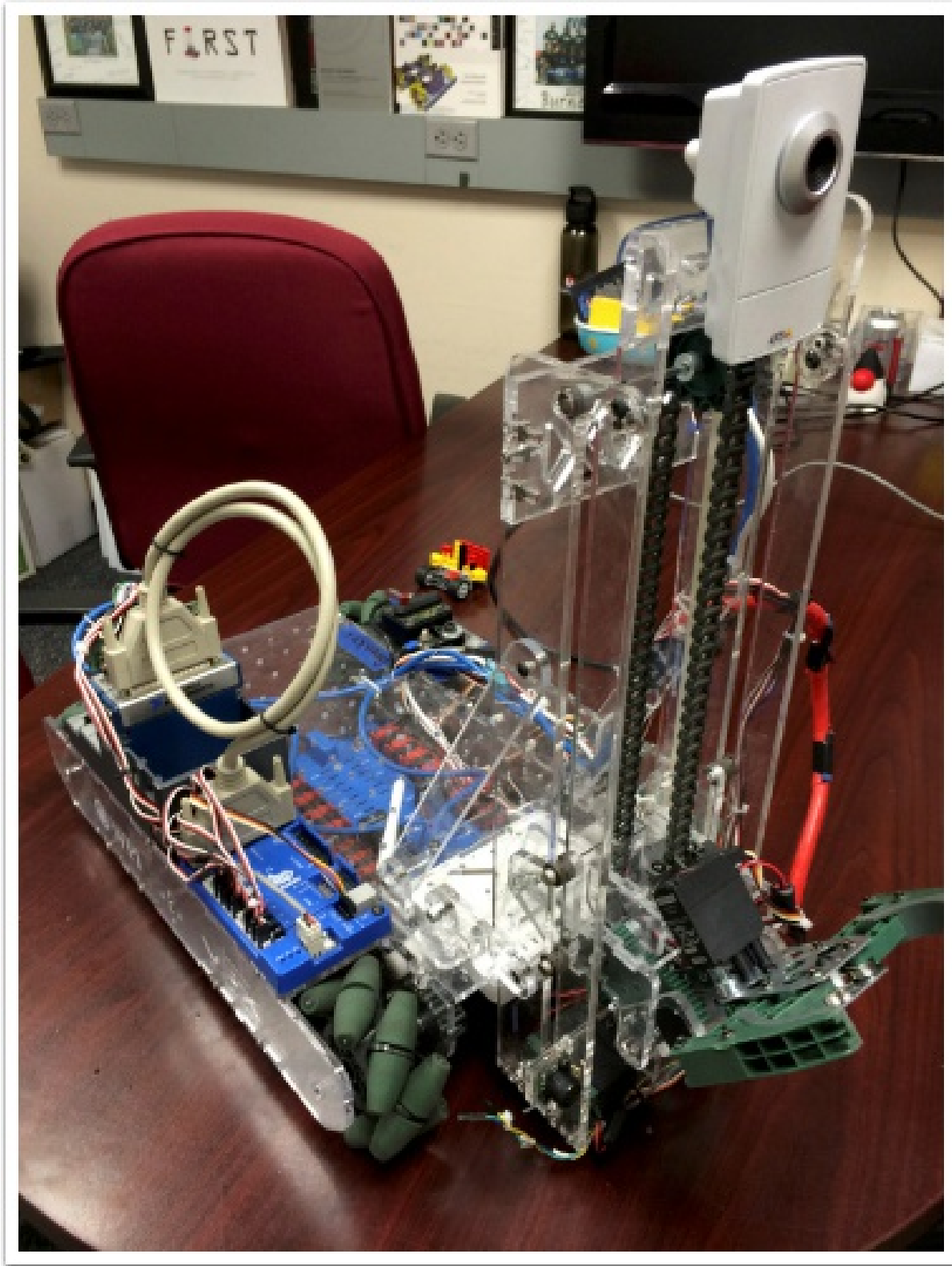
4.5.1 Drivetrains padrão

Robôs de acionamento diferencial



Essas bases de acionamento normalmente têm duas ou mais rodas on-line de tração ou omni por lado (por exemplo, 6WD ou 8WD) e também podem ser conhecidas como “skid-steer”, “tank drive” ou “West Coast Drive”. O sistema de transmissão do Kit de peças é um exemplo de acionamento diferencial. Essas unidades de transmissão são capazes de dirigir para frente / para trás e podem girar dirigindo os dois lados em direções opostas, fazendo com que as rodas deslizem para os lados. Essas unidades de transmissão não são capazes de movimento translacional lateral.

Mecanum Drive



Mecanum drive é um método de dirigir usando rodas especialmente projetadas que permitem

ao robô dirigir em qualquer direção sem alterar a orientação do robô. Um robô com um sistema de transmissão convencional (todas as rodas apontando na mesma direção) deve girar na direção que precisa dirigir. Um robô mecanum pode se mover em qualquer direção sem primeiro girar e é chamado de acionamento holonômico. As rodas (mostradas neste robô) possuem roletes que fazem com que as forças de acionamento sejam aplicadas em um ângulo de 45 graus em vez de para a frente, como no caso de um acionamento convencional.

Quando vistos de cima, os roletes no trem de força mecanum devem formar um padrão 'X'. Isso resulta nos vetores de força (ao dirigir a roda para frente) nas duas rodas dianteiras apontando para frente e para dentro e as duas rodas traseiras apontando para frente e para fora. Girando as rodas em direções diferentes, vários componentes dos vetores de força são cancelados, resultando no movimento desejado do robô. Um gráfico rápido de diferentes movimentos foi fornecido abaixo, traçando os vetores de força para cada um desses movimentos, podendo ajudar a entender como essas unidades de transmissão funcionam. Ao variar as velocidades das rodas, além da direção, os movimentos podem ser combinados, resultando em translação em qualquer direção e rotação, simultaneamente.

4.5.2 Convenções de classe de unidade

Note: Este artigo contém as convenções e padrões usados pelas classes WPILib Drive (DifferentialDrive, MecanumDrive e KilloughDrive). Para mais detalhes sobre o uso dessas classes, consulte os artigos subsequentes.

Inversão do motor

Por padrão, a classe inverte as saídas do motor para o lado direito do trem de força. Geralmente, isso significa que nenhuma inversão precisa ser feita nos objetos individuais do SpeedController. Para desativar esse comportamento, use o método `setRightSideInverted()`.

Squaring Inputs & Input Deadband

Ao dirigir robôs, geralmente é desejável manipular as entradas do joystick, de modo que o robô tenha um controle mais preciso em baixas velocidades enquanto ainda usa toda a faixa de saída. Uma maneira de fazer isso é alinhar ao quadrado a entrada do joystick e reaplicar o sinal. Por padrão, a classe Differential Drive quadrará as entradas. Se isso não for desejado (por exemplo, se você passar valores de um PIDController), use um dos métodos de unidade com o parâmetro `squaredInputs` e defina-o como `false`.

Por padrão, a classe Unidade Diferencial aplica uma faixa morta de entrada de 0,02. Isso significa que os valores de entrada com magnitude abaixo de 0,02 (após qualquer quadrado como descrito acima) serão definidos como 0. Na maioria dos casos, essas pequenas entradas resultam da centralização imperfeita do joystick e não são suficientes para causar movimento do trem de força, a faixa morta ajuda a reduzir desnecessários aquecimento do motor que pode resultar da aplicação desses pequenos valores no sistema de transmissão. Para alterar a banda morta, use o método `setDeadband()`.

Motor seguro

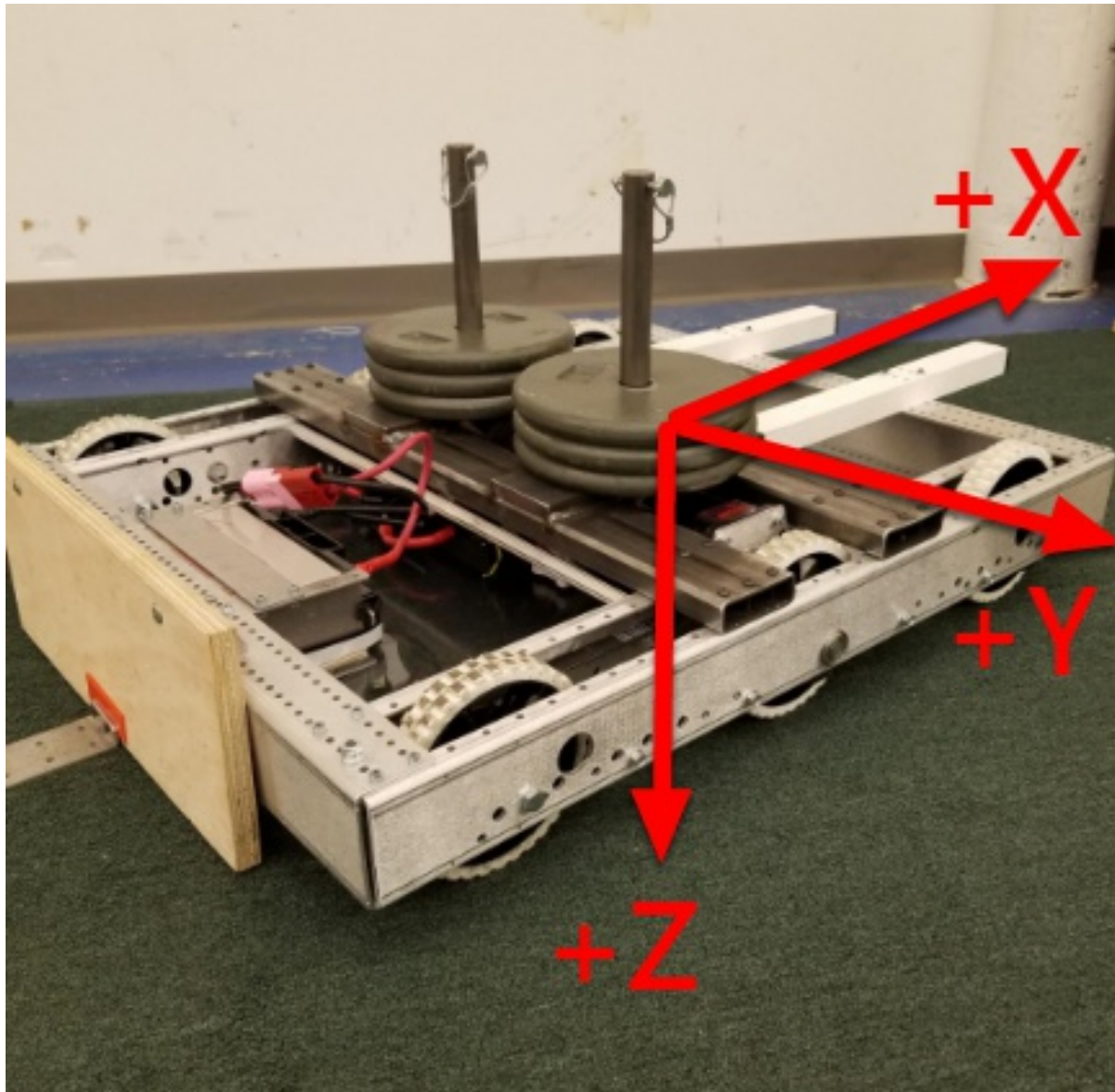
O Motor seguro é um mecanismo no WPILib que pega o conceito de um cão de guarda e o divide em um cão de guarda (temporizador de segurança do motor) para cada atuador individual. Observe que esse mecanismo de proteção é um complemento ao System Watchdog, que é controlado pelo código de Comunicações de Rede e pelo FPGA e desativará todas as saídas do atuador se ele não receber um pacote de dados válido por 125ms.

A finalidade do mecanismo de segurança do motor é a mesma de um cronômetro de vigilância, para desativar mecanismos que possam causar danos a si mesmos, pessoas ou propriedades se o código travar e não atualizar adequadamente a saída do atuador. A Segurança do motor quebra esse conceito por atuador, para que você possa determinar apropriadamente onde é necessário e onde não é. Exemplos de mecanismos que devem ter a segurança do motor ativada são sistemas como trens e braços de transmissão. Se esses sistemas ficarem presos a um valor específico, poderão causar danos ao meio ambiente ou a si próprios. Um exemplo de mecanismo que pode não precisar de segurança do motor é um volante giratório para um atirador. Se esse mecanismo ficar bloqueado em um valor específico, ele simplesmente continuará girando até que o robô seja desativado.

O recurso Segurança do motor opera mantendo um temporizador que rastreia quanto tempo faz desde que o método `feed()` foi chamado para esse atuador. O código na classe `Driver Station` inicia uma comparação desses temporizadores com os valores de tempo limite de qualquer atuador com segurança ativada a cada 5 pacotes recebidos (100 ms nominais). Os métodos `set()` de cada classe de controlador de velocidade e os métodos `set()` e `setAngle()` da classe de servo chamam `feed()` para indicar que a saída do atuador foi atualizada. The Motor Safety interface of speed controllers can be interacted with by the user using the following methods:

Por padrão, todos os objetos `RobotDrive` habilitam a Segurança do Motor. Dependendo do mecanismo e da estrutura do seu programa, você pode configurar o tempo limite da segurança do motor (em segundos). O tempo limite é configurado por atuador e não é uma configuração global. O valor padrão (e mínimo útil) é 100ms.

Convenções do Eixo



Esta biblioteca usa a convenção de eixos NED (North-East-Down como referência externa no quadro mundial). O eixo X positivo aponta para a frente, o eixo Y positivo aponta para a direita e o eixo Z positivo aponta para baixo. As rotações seguem a regra da direita, portanto, a rotação no sentido horário ao redor do eixo Z é positiva.

Warning: Esta convenção é diferente da convenção para joysticks que geralmente têm -Y como Up (geralmente mapeado para acelerar) e + X como Right. Preste muita atenção aos exemplos abaixo se desejar ajuda com o mapeamento típico de Joystick-> Drive.

4.5.3 Usando DifferentialDrive para controlar o comando diferencial (CMB) dos robôs

Note: O WPILib fornece classes de movimentação de robôs separadas para as configurações mais comuns de trem de força (diferencial, mecanismo e Killough). A classe DifferentialDrive lida com a configuração do drivetrain diferencial. Essas bases de acionamento normalmente têm duas ou mais rodas on-line de tração ou omni por lado (por exemplo, 6WD ou 8WD) e também podem ser conhecidas como “skid-steer”, “tank drive” ou “West Coast Drive”. O sistema de transmissão do Kit de peças é um exemplo de acionamento diferencial. Existem métodos para controlar a unidade com 3 estilos diferentes (“Tanque”, “Arcade” ou “Curvatura”), explicados no artigo abaixo.

Unidade Multi-Motor com SpeedControllerGroups

Muitas unidades de transmissão FRC têm mais de 1 motor em cada lado. Para usá-los com o DifferentialDrive, os motores de cada lado precisam ser coletados em um único SpeedController, usando a classe SpeedControllerGroup. Os exemplos abaixo mostram um trem de força com 4 motores (2 por lado). Para estender a mais motores, basta criar os controladores adicionais e passar todos eles para o construtor de grupo SpeedController (é necessário um número arbitrário de entradas).

Modos de acionamento

Note: A classe DifferentialDrive contém três modos padrão diferentes de acionar os motores do seu robô.

- Tank Drive, que controla os lados esquerdo e direito de forma independente;
 - Arcade Drive, que controla a velocidade de avanço e rotação;
 - Curvature Drive, um subconjunto do Arcade Drive, que faz com que o seu robô lide com um carro com curvas constantes.
-

Como mencionado acima, a classe DifferentialDrive contém três métodos padrão para controlar robôs skid-steer ou WCD. Observe que você pode criar seus próprios métodos para controlar a direção do robô e pedir que chamem tankDrive () com as entradas derivadas dos motores esquerdo e direito.

O modo de acionamento do tanque é usado para controlar cada lado do trem de força de forma independente (geralmente com um eixo de joystick individual controlando cada um). Este exemplo mostra como usar o eixo Y de dois joysticks separados para executar o drivetrain no modo Tank. A construção dos objetos foi omitida, acima para a construção do trem de força e aqui para a construção do Joystick.

O modo Arcade Drive é usado para controlar o trem de força usando velocidade / acelerador e taxa de rotação. Isso geralmente é usado com dois eixos de um único joystick ou dividido entre joysticks (geralmente em um único gamepad), com o acelerador saindo de um manípulo e a rotação de outro. Este exemplo mostra como usar um único joystick com o modo Arcade. A construção dos objetos foi omitida, acima para a construção do trem de força e aqui para a construção do Joystick.

Como o Arcade Drive, o modo Curvature Drive é usado para controlar o trem de força usando velocidade / aceleração e taxa de rotação. A diferença é que o controle de rotação está tentando controlar o raio de curvatura em vez da taxa de mudança de rumo. Este modo também possui um parâmetro de rotação rápida que é usado para ativar um submodo que permite a rotação no lugar. Este exemplo mostra como usar um único joystick com o modo Curvatura. A construção dos objetos foi omitida, acima para a construção do trem de força e aqui para a construção do Joystick.

4.6 Movimento repetitivo de baixa potência - servos de controle com WPILib

Os Servo Motors são um tipo de motores que integram feedback posicional ao motor, a fim de permitir que um único motor execute movimentos repetíveis e controláveis, assumindo a posição de sinal de entrada. O WPILib fornece a capacidade de controlar servos que correspondem à especificação de entrada de “hábito” comum (sinal PWM, largura de pulso de 1,0ms-2,0ms).

4.7 LEDs

Os LEDs são comumente usados pelas equipes há vários anos por vários motivos. Eles permitem que as equipes depurem a funcionalidade do robô do público, forneçam um marcador visual para o robô e podem simplesmente adicionar algum apelo visual. O WPILib possui uma API para controlar os LEDs WS2812 com seus pinos de dados conectados via PWM.

4.7.1 Referenciando os LEDs

Você, primeiramente, cria um `AddressableLED` objeto que usa a porta PWM como argumento. Ele *must* ser um cabeçalho PWM no roboRIO. Depois, você define o número de LEDs localizados na sua faixa de LEDs, o que pode ser feito com a `setLength()` função.

Important: É importante notar que definir o comprimento do cabeçalho LED é uma tarefa cara e ele não recomendado para executar este periodicamente.

Depois que o comprimento da faixa for definido, você precisará criar um `AddressableLEDBuffer` objeto que recebe o número de LEDs como entrada. Você ligará `myAddressableLed.setData(myAddressableLEDBuffer)` para definir os dados de saída do led. Finalmente, você pode ligar to set the led output data `myAddressableLed.start()` para escrever a saída continuamente. Abaixo está um exemplo completo do processo de inicialização.

Note: C++ não possui um `AddressableLEDBuffer`, e usa um `Array`.

4.7.2 Definindo a faixa para uma cor

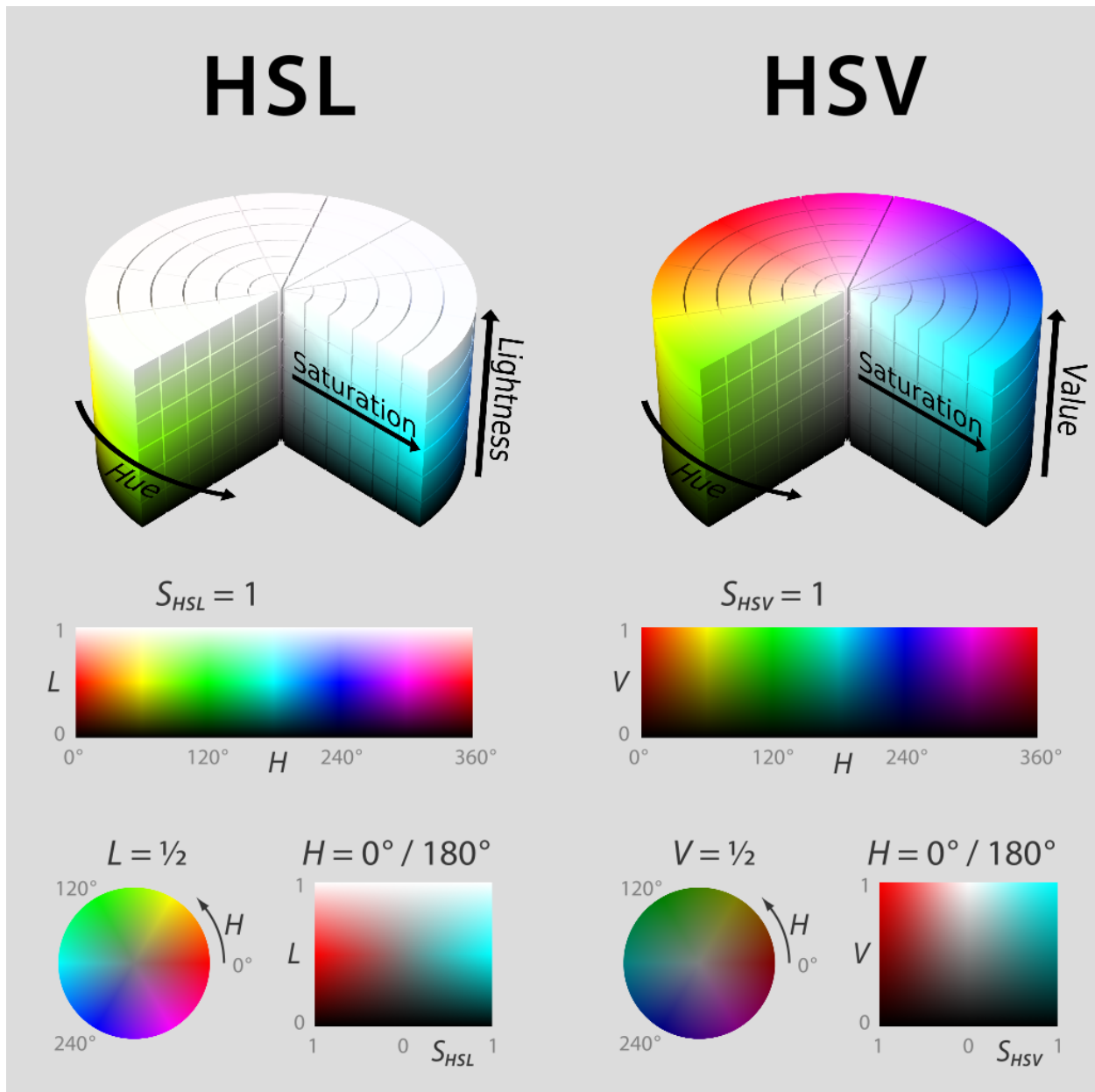
A cor pode ser definida como um led individual na faixa usando dois métodos: “`setRGB`”, que aceita valores RGB como entrada e `setHSV()` na qual aceita valores HSV como entrada.

Usando valores RGB

RGB significa vermelho, verde e azul. Este é um modelo de cores bastante comum, pois é bastante fácil de entender. Os LEDs podem ser configurados com o `setRGB` um método que leva 4 argumentos: índice do LED, quantidade de vermelho, quantidade de verde, quantidade de azul. A quantidade de vermelho, verde e azul são valores inteiros entre 0 e 255.

Usando valores HSV

HSV significa Matiz, Saturação e Valor. Matiz descreve a cor ou matiz, saturação sendo a quantidade de cinza e valor sendo o brilho. No WPILib, Hue é um número inteiro de 0 a 180. Saturação e Valor são números inteiros de 0 a 255. Se você observar um seletor de cores como [Google's](#), a matiz será de 0 a 360 e a saturação e o valor variam de 0% a 100%. É da mesma maneira que o OpenCV lida com cores HSV. Verifique se os valores HSV inseridos no WPILib estão corretos ou se a cor produzida pode não ser a mesma esperada.



Os LEDs podem ser configurados com o “setHSV” método que utiliza 4 argumentos: índice do LED, matiz, saturação e valor. Um exemplo é mostrado abaixo para definir a cor de uma faixa de LED para vermelho (matiz de 0).

4.7.3 Criando um efeito arco-íris

O método abaixo faz algumas coisas importantes. Dentro do loop *for*, distribui igualmente o matiz por todo o comprimento do fio e armazena o matiz de LED individual em uma variável chamada *hue*. Em seguida, o loop *for* define o valor HSV desse pixel especificado usando o valor *hue*.

Movendo-se para fora do loop *for*, o `m_rainbowFirstPixelHue` itera o pixel que contém o matiz “inicial”, criando o efeito arco-íris. `m_rainbowFirstPixelHue` para verifica se a matiz

está dentro dos limites da matiz de 180. Isso ocorre porque a matiz HSV é um valor de 0 a 180.

Note: É uma boa prática de robô manter o `robotPeriodic()` método o mais limpo possível, por isso, criaremos um método para lidar com a configuração de nossos dados de LED. Poderemos ligar o método `rainbow()` e ligar para `robotPeriodic()`.

Agora que tem-se o nosso `rainbow` método criado, vamos ter que, na realidade, ligar o método e definir os dados do LED.

5.1 Sensor Overview - Software

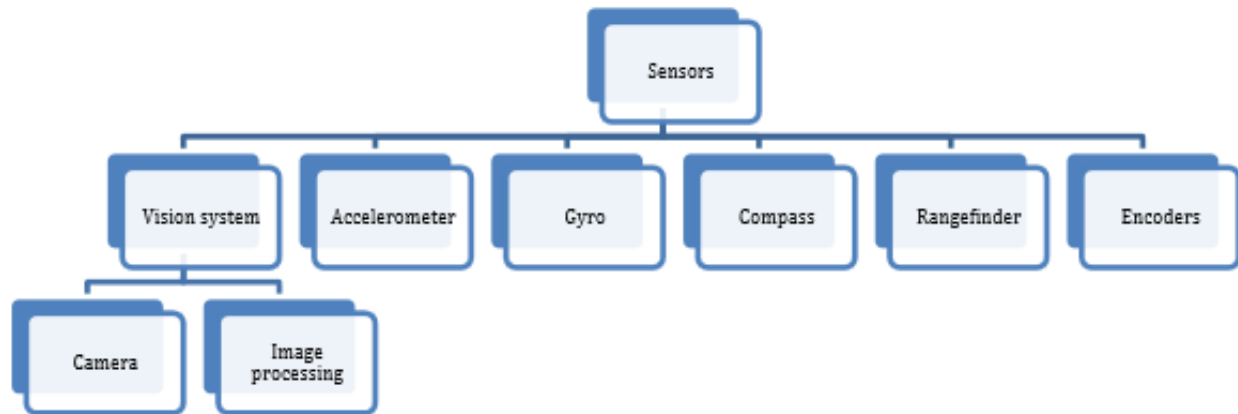
Note: This section covers using sensors in software. For a guide to sensor hardware, see [docs/hardware/sensors/sensor-overview-hardware:Sensor Overview - Hardware](#).

Note: While cameras may definitely be considered “sensors”, vision processing is a sufficiently-complicated subject that it is covered in its own section, rather than here.

In order to be effective, it is often vital for robots to be able to gather information about their surroundings. Devices that provide feedback to the robot on the state of its environment are called “sensors.” WPILib innately supports a large variety of sensors through classes included in the library. This section will provide a guide to both using common sensor types through WPILib, as well as writing code for sensors without official support.

5.1.1 What sensors does WPILIB support?

The roboRIO includes a [FPGA](#) which allows accurate real-time measuring of a variety of sensor input. WPILib, in turn, provides a number of classes for accessing this functionality.



WPILib provides native support for:

- Accelerometers
- *Gyroscopes*
- *Ultrasonic rangefinders*
- Potentiometers
- Counters
- *Quadrature encoders*
- *Limit switches*

Additionally, WPILib includes lower-level classes for interfacing directly with the FPGA's digital and analog inputs and outputs.

5.2 Acelerômetros - Software

Note: Esta seção abrange acelerômetros em software. Para obter um guia de hardware para acelerômetros, consulte docs/hardware/sensors/accelerometers-hardware:Accelerometers - Hardware.

Um acelerômetro é um dispositivo que mede a aceleração.

Os acelerômetros geralmente vêm em dois tipos: eixo único e 3 eixos. Um acelerômetro de eixo único mede a aceleração ao longo de uma dimensão espacial; um acelerômetro de 3 eixos mede a aceleração ao longo das três dimensões espaciais de uma só vez.

O WPILib suporta acelerômetros de eixo único através da '**AnalogAccelerometer**' .

Os acelerômetros de três eixos geralmente exigem protocolos de comunicação mais complicados (como SPI ou I2C) para enviar dados multidimensionais. O WPILib possui suporte nativo para os seguintes acelerômetros de 3 eixos: - *ADXL345_I2C* - *ADXL345_SPI* - *ADXL362* - *BuiltInAccelerometer*

5.2.1 Acelerômetro analógico

The AnalogAccelerometer (Java, C++) permite que os usuários leiam valores de um acelerômetro de eixo único conectado a uma das entradas analógicas do roboRIO.

Se os usuários tiverem um acelerômetro analógico de 3 eixos, poderão usar três instâncias dessa classe, uma para cada eixo.

5.2.2 A interface do acelerômetro

Todos os acelerômetros de 3 eixos no WPILib implementam a Accelerometer interface (Java, C++). Essa interface define a funcionalidade e as configurações comuns a todos os acelerômetros de 3 eixos suportados.

O Accelerometer interface contém getters para a aceleração ao longo de cada direção cardinal (x, y e z), bem como um setter para o intervalo de acelerações que o acelerômetro medirá.

Warning: Nem todos os acelerômetros são capazes de medir todas as faixas.

ADXL345_I2C

A ADXL345_I2C (Java, C++) fornece suporte para o acelerômetro ADXL345 através do barramento de comunicação I2C.

ADXL345_SPI

A ADXL345_SPI (Java, C++) fornece suporte para o acelerômetro ADXL345 por meio do barramento de comunicações SPI communications.

ADXL362

A ADXL362 (Java, C++) fornece suporte para o acelerômetro ADXL362 através do barramento de comunicações SPI.

BuiltInAccelerometer

A BuiltInAccelerometer (Java, C++) fornece acesso ao próprio acelerômetro interno do roboRIO:

5.2.3 Third-party accelerometers

Embora o WPILib forneça suporte nativo para vários acelerômetros disponíveis no kit de peças ou através da FIRST Choice, existem alguns dispositivos populares AHRS (Sistema de Referência de Atitude e Direção) comumente usados no FRC que incluem acelerômetros. Eles geralmente são controlados por meio de bibliotecas de fornecedores, embora, se tiverem uma saída analógica simples, possam ser usados com **'AnalogAccelerometer'**

5.2.4 Usando acelerômetros no código

Note: Acelerômetros, como o próprio nome sugere, medem a aceleração. Acelerômetros precisos podem ser usados para determinar a posição através da dupla integração (já que a aceleração é a segunda derivada da posição), da mesma maneira que os giroscópios são usados para determinar a direção. No entanto, os acelerômetros disponíveis para uso em FRC não têm qualidade suficientemente alta para serem usados dessa maneira.

Recomenda-se o uso de acelerômetros no FRC para qualquer aplicação que precise de uma medição aproximada da aceleração atual. Isso pode incluir a detecção de colisões com outros robôs ou elementos de campo, para que os mecanismos vulneráveis possam ser retraídos automaticamente. Eles também podem ser usados para determinar quando o robô está passando por terrenos acidentados para uma rotina autônoma (como atravessar as defesas no FIRST Stronghold).

Para detectar colisões, geralmente é mais robusto medir o empurrão do que a aceleração. O empurrão é a derivada (ou taxa de mudança) da aceleração e indica a rapidez com que as forças do robô estão mudando - o impulso repentino de uma colisão causa um aumento acentuado no empurrão. Jerk pode ser determinado simplesmente tomando a diferença das medições de aceleração subsequentes e dividindo pelo tempo entre elas:

A maioria dos acelerômetros legais para o uso de FRC é bastante barulhenta e geralmente é uma boa ideia combiná-los com a `LinearFilter` ([Java](#), [C++](#)) para reduzir o ruído.

5.3 Gyroscopes - Software

Note: This section covers gyros in software. For a hardware guide to gyros, see [docs/hardware/sensors/gyros-hardware:Gyroscopes - Hardware](#).

A gyroscope, or “gyro,” is an angular rate sensor typically used in robotics to measure and/or stabilize robot headings. WPILib natively provides specific support for the ADXRS450 gyro available in the kit of parts, as well as more general support for a wider variety of analog gyros through the [AnalogGyro](#) class.

5.3.1 The Gyro interface

All natively-supported gyro objects in WPILib implement the Gyro interface ([Java](#), [C++](#)). This interface provides methods for getting the current angular rate and heading, zeroing the current heading, and calibrating the gyro.

Note: It is crucial that the robot remain stationary while calibrating a gyro.

ADXRS450_Gyro

The `ADXRS450_Gyro` class ([Java](#), [C++](#)) provides support for the Analog Devices ADXRS450 gyro available in the kit of parts, which connects over the SPI bus.

Note: ADXRS450 Gyro accumulation is handled through special circuitry in the FPGA; accordingly only a single instance of ADXRS450_Gyro may be used.

Java

```
// Creates an ADXRS450_Gyro object on the MXP SPI port
Gyro gyro = new ADXRS450_Gyro(SPI.Port.kMXP);
```

C++

```
// Creates an ADXRS450_Gyro object on the MXP SPI port
ADXRS450_Gyro gyro{SPI::Port::kMXP};
```

AnalogGyro

The AnalogGyro class (Java, C++) provides support for any single-axis gyro with an analog output.

Note: Gyro accumulation is handled through special circuitry in the FPGA; accordingly, AnalogGyro's may only be used on analog ports 0 and 1.

Java

```
// Creates an AnalogGyro object on port 0
Gyro gyro = new AnalogGyro(0);
```

C++

```
// Creates an AnalogGyro object on port 0
AnalogGyro gyro{0};
```

5.3.2 Third-party gyros

While WPILib provides native support for a the ADXRS450 gyro available in the kit of parts and for any analog gyro, there are a few popular AHRS (Attitude and Heading Reference System) devices commonly used in FRC that include accelerometers and require more complicated communications. These are generally controlled through vendor libraries.

5.3.3 Using gyros in code

Note: As gyros measure rate rather than position, position is inferred by integrating (adding up) the rate signal to get the total change in angle. Thus, gyro angle measurements are always relative to some arbitrary zero angle (determined by the angle of the gyro when either the robot was turned on or a zeroing method was called), and are also subject to accumulated errors (called “drift”) that increase in magnitude the longer the gyro is used. The amount of drift varies with the type of gyro.

Gyros are extremely useful in FRC for both measuring and controlling robot heading. Since FRC matches are generally short, total gyro drift over the course of an FRC match tends to be manageably small (on the order of a couple of degrees for a good-quality gyro). Moreover, not all useful gyro applications require the absolute heading measurement to remain accurate over the course of the entire match.

Displaying the robot heading on the dashboard

Shuffleboard includes a widget for displaying heading data from a Gyro in the form of a compass. This can be helpful for viewing the robot heading when sight lines to the robot are obscured:

Java

```
Gyro gyro = new ADXRS450_Gyro(SPI.Port.kMXP);

public void robotInit() {
    // Places a compass indicator for the gyro heading on the dashboard
    // Explicit down-cast required because Gyro does not extend Sendable
    Shuffleboard.getTab("Example tab").add((Sendable) gyro);
}
```

C++

```
frc::ADXRS450_Gyro gyro{frc::SPI::Port::kMXP};

void Robot::RobotInit() {
    // Places a compass indicator for the gyro heading on the dashboard
    frc::Shuffleboard.GetTab("Example tab").Add(gyro);
}
```

Stabilizing heading while driving

A very common use for a gyro is to stabilize robot heading while driving, so that the robot drives straight. This is especially important for holonomic drives such as mecanum and swerve, but is extremely useful for tank drives as well.

This is typically achieved by closing a PID controller on either the turn rate or the heading, and piping the output of the loop to one's turning control (for a tank drive, this would be a speed differential between the two sides of the drive).

Warning: Like with all control loops, users should be careful to ensure that the sensor direction and the turning direction are consistent. If they are not, the loop will be unstable and the robot will turn wildly.

Example: Tank drive stabilization using turn rate

The following example shows how to stabilize heading using a simple P loop closed on the turn rate. Since a robot that is not turning should have a turn rate of zero, the setpoint for the loop is implicitly zero, making this method very simple.

Java

```

Gyro gyro = new ADXRS450_Gyro(SPI.Port.kMXP);

// The gain for a simple P loop
double kP = 1;

// Initialize motor controllers and drive
Spark left1 = new Spark(0);
Spark left2 = new Spark(1);

Spark right1 = new Spark(2);
Spark right2 = new Spark(3);

SpeedControllerGroup leftMotors = new SpeedControllerGroup(left1, left2);
SpeedControllerGroup rightMotors = new SpeedControllerGroup(right1, right2);

DifferentialDrive drive = new DifferentialDrive(leftMotors, rightMotors);

@Override
public void autonomousPeriodic() {
    // Setpoint is implicitly 0, since we don't want the heading to change
    double error = -gyro.getRate();

    // Drives forward continuously at half speed, using the gyro to stabilize the
    ↪ heading
    drive.tankDrive(.5 + kP * error, .5 - kP * error);
}

```

C++

```

frc::ADXRS450_Gyro gyro{frc::SPI::Port::kMXP};

// The gain for a simple P loop
double kP = 1;

// Initialize motor controllers and drive
frc::Spark left1{0};
frc::Spark left2{1};
frc::Spark right1{2};
frc::Spark right2{3};

frc::SpeedControllerGroup leftMotors{left1, left2};
frc::SpeedControllerGroup rightMotors{right1, right2};

frc::DifferentialDrive drive{leftMotors, rightMotors};

void Robot::AutonomousPeriodic() {
    // Setpoint is implicitly 0, since we don't want the heading to change
    double error = -gyro.GetRate();

    // Drives forward continuously at half speed, using the gyro to stabilize the
    ↪ heading
    drive.TankDrive(.5 + kP * error, .5 - kP * error);
}

```

More-advanced implementations can use a more-complicated control loop. When closing the loop on the turn rate for heading stabilization, PI loops are particularly effective.

Example: Tank drive stabilization using heading

The following example shows how to stabilize heading using a simple P loop closed on the heading. Unlike in the turn rate example, we will need to set the setpoint to the current heading before starting motion, making this method slightly more-complicated.

Java

```
Gyro gyro = new ADXRS450_Gyro(SPI.Port.kMXP);

// The gain for a simple P loop
double kP = 1;

// The heading of the robot when starting the motion
double heading;

// Initialize motor controllers and drive
Spark left1 = new Spark(0);
Spark left2 = new Spark(1);

Spark right1 = new Spark(2);
Spark right2 = new Spark(3);

SpeedControllerGroup leftMotors = new SpeedControllerGroup(left1, left2);
SpeedControllerGroup rightMotors = new SpeedControllerGroup(right1, right2);

DifferentialDrive drive = new DifferentialDrive(leftMotors, rightMotors);

@Override
public void autonomousInit() {
    // Set setpoint to current heading at start of auto
    heading = gyro.getAngle();
}

@Override
public void autonomousPeriodic() {
    double error = heading - gyro.getAngle();

    // Drives forward continuously at half speed, using the gyro to stabilize the
    // heading
    drive.tankDrive(.5 + kP * error, .5 - kP * error);
}
```

C++

```
frc::ADXRS450_Gyro gyro{frc::SPI::Port::kMXP};

// The gain for a simple P loop
double kP = 1;

// The heading of the robot when starting the motion
double heading;

// Initialize motor controllers and drive
frc::Spark left1{0};
frc::Spark left2{1};
frc::Spark right1{2};
frc::Spark right2{3};
```

(continues on next page)

(continued from previous page)

```

frc::SpeedControllerGroup leftMotors{left1, left2};
frc::SpeedControllerGroup rightMotors{right1, right2};

frc::DifferentialDrive drive{leftMotors, rightMotors};

void Robot::AutonomousInit() {
    // Set setpoint to current heading at start of auto
    heading = gyro.GetAngle();
}

void Robot::AutonomousPeriodic() {
    double error = heading - gyro.GetAngle();

    // Drives forward continuously at half speed, using the gyro to stabilize the
    // heading
    drive.TankDrive(.5 + kP * error, .5 - kP * error);
}

```

More-advanced implementations can use a more-complicated control loop. When closing the loop on the heading for heading stabilization, PD loops are particularly effective.

Turning to a set heading

Another common and highly-useful application for a gyro is turning a robot to face a specified direction. This can be a component of an autonomous driving routine, or can be used during teleoperated control to help align a robot with field elements.

Much like with heading stabilization, this is often accomplished with a PID loop - unlike with stabilization, however, the loop can only be closed on the heading. The following example code will turn the robot to face 90 degrees with a simple P loop:

Java

```

Gyro gyro = new ADXRS450_Gyro(SPI.Port.kMXP);

// The gain for a simple P loop
double kP = 1;

// Initialize motor controllers and drive
Spark left1 = new Spark(0);
Spark left2 = new Spark(1);

Spark right1 = new Spark(2);
Spark right2 = new Spark(3);

SpeedControllerGroup leftMotors = new SpeedControllerGroup(left1, left2);
SpeedControllerGroup rightMotors = new SpeedControllerGroup(right1, right2);

DifferentialDrive drive = new DifferentialDrive(leftMotors, rightMotors);

@Override
public void autonomousPeriodic() {
    // Find the heading error; setpoint is 90
    double error = 90 - gyro.getAngle();
}

```

(continues on next page)

(continued from previous page)

```
// Turns the robot to face the desired direction
drive.tankDrive(kP * error, kP * error);
}
```

C++

```
frc::ADXRS450_Gyro gyro{frc::SPI::Port::kMXP};

// The gain for a simple P loop
double kP = 1;

// Initialize motor controllers and drive
frc::Spark left1{0};
frc::Spark left2{1};
frc::Spark right1{2};
frc::Spark right2{3};

frc::SpeedControllerGroup leftMotors{left1, left2};
frc::SpeedControllerGroup rightMotors{right1, right2};

frc::DifferentialDrive drive{leftMotors, rightMotors};

void Robot::AutonomousPeriodic() {
    // Find the heading error; setpoint is 90
    double error = 90 - gyro.GetAngle();

    // Turns the robot to face the desired direction
    drive.TankDrive(kP * error, kP * error);
}
```

As before, more-advanced implementations can use more-complicated control loops.

Note: Turn-to-angle loops can be tricky to tune correctly due to static friction in the drive-train, especially if a simple P loop is used. There are a number of ways to account for this; one of the most common/effective is to add a “minimum output” to the output of the control loop. Another effective strategy is to cascade to well-tuned velocity controllers on each side of the drive.

5.4 Ultrasonics - Software

Note: This section covers ultrasonics in software. For a hardware guide to ultrasonics, see docs/hardware/sensors/ultrasonics-hardware:Ultrasonics - Hardware.

An ultrasonic sensor is commonly used to measure distance to an object using high-frequency sound. Generally, ultrasonics measure the distance to the closest object within their “field of view.”

There are two primary types of ultrasonics supported natively by WPILib:

- *Ping-response ultrasonics*

- *Analog ultrasonics*

5.4.1 Ping-response ultrasonics

The Ultrasonic class (Java, C++) provides support for ping-response ultrasonics. As ping-response ultrasonics (per the: name) require separate pins for both sending the ping and measuring the response, users must specify DIO pin numbers for both output and input when constructing an Ultrasonic instance:

Java

```
// Creates a ping-response Ultrasonic object on DIO 1 and 2.
Ultrasonic ultrasonic = new Ultrasonic(1, 2);
```

C++

```
// Creates a ping-response Ultrasonic object on DIO 1 and 2.
frc::Ultrasonic ultrasonic{1, 2};
```

It is highly recommended to use ping-response ultrasonics in “automatic mode,” as this will allow WPILib to ensure that multiple sensors do not interfere with each other:

Java

```
// Starts the ultrasonic sensor running in automatic mode
ultrasonic.setAutomaticMode(true);
```

C++

```
// Starts the ultrasonic sensor running in automatic mode
ultrasonic.SetAutomaticMode(true);
```

5.4.2 Analog ultrasonics

Some ultrasonic sensors simply return an analog voltage corresponding to the measured distance. These sensors can may simply be used with the *AnalogPotentiometer* class.

5.4.3 Third-party ultrasonics

Other ultrasonic sensors offered by third-parties may use more complicated communications protocols (such as I2C or SPI). WPILib does not provide native support for any such ultrasonics; they will typically be controlled with vendor libraries.

5.4.4 Using ultrasonics in code

Ultrasonic sensors are very useful for determining spacing during autonomous routines. For example, the following code will drive the robot forward until the ultrasonic measures a distance of 12 inches to the nearest object, and then stop:

Java

```
// Creates a ping-response Ultrasonic object on DIO 1 and 2.
Ultrasonic ultrasonic = new Ultrasonic(1, 2);

// Initialize motor controllers and drive
Spark left1 new Spark(0);
Spark left2 = new Spark(1);

Spark right1 = new Spark(2);
Spark right2 = new Spark(3);

SpeedControllerGroup leftMotors = new SpeedControllerGroup(left1, left2);
SpeedControllerGroup rightMotors = new SpeedControllerGroup(right1, right2);

DifferentialDrive drive = new DifferentialDrive(leftMotors, rightMotors);

@Override
public void robotInit() {
    // Start the ultrasonic in automatic mode
    ultrasonic.setAutomaticMode(true);
}

@Override
public void autonomousPeriodic() {
    if(ultrasonic.GetRangeInches() > 12) {
        drive.tankDrive(.5, .5);
    }
    else {
        drive.tankDrive(0, 0);
    }
}
```

C++

```
// Creates a ping-response Ultrasonic object on DIO 1 and 2.
frc::Ultrasonic ultrasonic{1, 2};

// Initialize motor controllers and drive
frc::Spark left1{0};
frc::Spark left2{1};
frc::Spark right1{2};
frc::Spark right2{3};

frc::SpeedControllerGroup leftMotors{left1, left2};
frc::SpeedControllerGroup rightMotors{right1, right2};

frc::DifferentialDrive drive{leftMotors, rightMotors};

void Robot::RobotInit() {
    // Start the ultrasonic in automatic mode
    ultrasonic.SetAutomaticMode(true);
}

void Robot::AutonomousPeriodic() {
    if(ultrasonic.GetRangeInches() > 12) {
        drive.TankDrive(.5, .5);
    }
    else {
```

(continues on next page)

(continued from previous page)

```

        drive.TankDrive(0, 0);
    }
}

```

Additionally, ping-response ultrasonics can be sent to Shuffleboard, where they will be displayed with their own widgets:

Java

```

// Creates a ping-response Ultrasonic object on DIO 1 and 2.
Ultrasonic ultrasonic = new Ultrasonic(1, 2);

public void robotInit() {
    // Places a the ultrasonic on the dashboard
    Shuffleboard.getTab("Example tab").add(ultrasonic);
}

```

C++

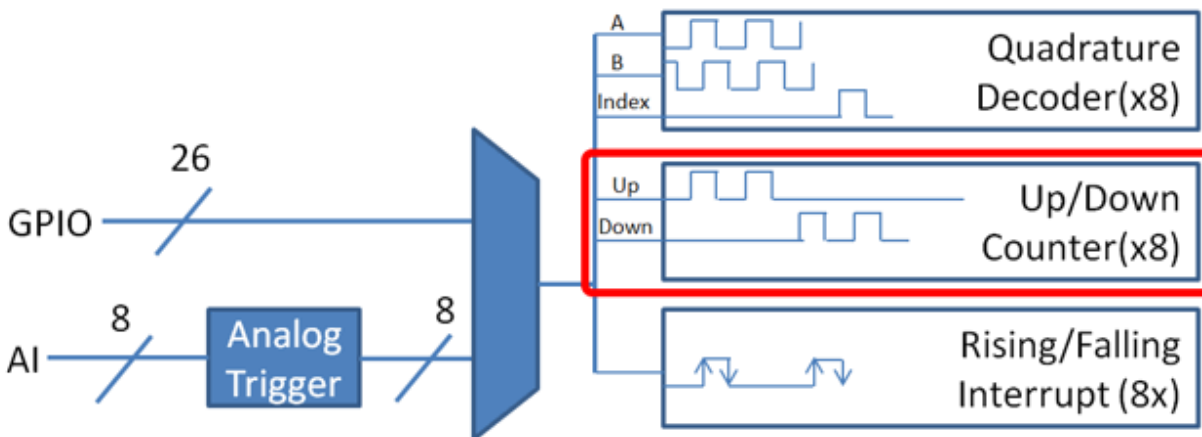
```

// Creates a ping-response Ultrasonic object on DIO 1 and 2.
frc::Ultrasonic ultrasonic{1, 2};

void Robot::RobotInit() {
    // Places the ultrasonic on the dashboard
    frc::Shuffleboard.GetTab("Example tab").Add(ultrasonic);
}

```

5.5 Contadores



A Counter classe (Java, C++) é uma classe versátil que permite a contagem de bordas de pulso em uma entrada digital. Counter é usado como um componente em várias classes WPILib mais complicadas (como *Encoder* e *Ultrasonic*), mas também é bastante útil por si só.

Note: There are a total of 8 counter units in the roboRIO FPGA, meaning no more than 8 Counter objects may be instantiated at any one time, including those contained as resources in other WPILib objects. For detailed information on when a Counter may be used by another

object, refer to the official API documentation.

5.5.1 Configuring a counter

The Counter class can be configured in a number of ways to provide differing functionalities.

Counter Modes

The Counter object may be configured to operate in one of four different modes:

1. *Two-pulse mode*: Counts up and down based on the edges of two different channels.
2. *Semi-period mode*: Measures the duration of a pulse on a single channel.
3. *Pulse-length mode*: Counts up and down based on the edges of one channel, with the direction determined by the duration of the pulse on that channel.
4. *External direction mode*: Counts up and down based on the edges of one channel, with a separate channel specifying the direction.

Note: In all modes except semi-period mode, the counter can be configured to increment either once per edge (2X decoding), or once per pulse (1X decoding). By default, counters are set to two-pulse mode, if only one channel is specified, the counter will only count up.

Two-pulse mode

In two-pulse mode, the Counter will count up for every edge/pulse on the specified “up channel,” and down for every edge/pulse on the specified “down channel.” A counter can be initialized in two-pulse with the following code:

Java

```
// Create a new Counter object in two-pulse mode
Counter counter = new Counter(Counter.Mode.k2Pulse);

@Override
public void robotInit() {
    // Set up the input channels for the counter
    counter.setUpSource(1);
    counter.setDownSource(2);

    // Set the decoding type to 2X
    counter.setUpSourceEdge(true, true);
    counter.setDownSourceEdge(true, true);
}
```

C++

```
// Create a new Counter object in two-pulse mode
frc::Counter counter{frc::Counter::Mode::k2Pulse};
```

(continues on next page)

(continued from previous page)

```

void Robot::RobotInit() {
    // Set up the input channels for the counter
    counter.SetupSource(1);
    counter.SetDownSource(2);

    // Set the decoding type to 2X
    counter.SetupSourceEdge(true, true);
    counter.SetDownSourceEdge(true, true);

```

Semi-period mode

In semi-period mode, the Counter will count the duration of the pulses on a channel, either from a rising edge to the next falling edge, or from a falling edge to the next rising edge. A counter can be initialized in semi-period mode with the following code:

Java

```

// Create a new Counter object in two-pulse mode
Counter counter = new Counter(Counter.Mode.kSemiperiod);

@Override
public void robotInit() {
    // Set up the input channel for the counter
    counter.setupSource(1);

    // Set the encoder to count pulse duration from rising edge to falling edge
    counter.setSemiPeriodMode(true);
}

```

C++

```

// Create a new Counter object in two-pulse mode
frc::Counter counter{frc::Counter::Mode::kSemiperiod};

void Robot() {
    // Set up the input channel for the counter
    counter.SetupSource(1);

    // Set the encoder to count pulse duration from rising edge to falling edge
    counter.SetSemiPeriodMode(true);
}

```

To get the pulse width, call the `getPeriod()` method:

Java

```

// Return the measured pulse width in seconds
counter.GetPeriod();

```

C++

```

// Return the measured pulse width in seconds
counter.getPeriod();

```

Pulse-length mode

In pulse-length mode, the counter will count either up or down depending on the length of the pulse. A pulse below the specified threshold time will be interpreted as a forward count and a pulse above the threshold is a reverse count. This is useful for some gear tooth sensors which encode direction in this manner. A counter can be initialized in this mode as follows:

Java

```
// Create a new Counter object in two-pulse mode
Counter counter = new Counter(Counter.Mode.kPulseLength);

@Override
public void robotInit() {
    // Set up the input channel for the counter
    counter.setUpSource(1);

    // Set the decoding type to 2X
    counter.setUpSourceEdge(true, true);

    // Set the counter to count down if the pulses are longer than .05 seconds
    counter.SetPulseLengthMode(.05)
}
```

C++

```
// Create a new Counter object in two-pulse mode
frc::Counter counter{frc::Counter::Mode::kPulseLength};

void Robot::RobotInit() {
    // Set up the input channel for the counter
    counter.SetUpSource(1);

    // Set the decoding type to 2X
    counter.SetUpSourceEdge(true, true);

    // Set the counter to count down if the pulses are longer than .05 seconds
    counter.setPulseLengthMode(.05)
```

External direction mode

In external direction mode, the counter counts either up or down depending on the level on the second channel. If the direction source is low, the counter will increase, if the direction source is high, the counter will decrease (to reverse this, see the next section). A counter can be initialized in this mode as follows:

Java

```
// Create a new Counter object in two-pulse mode
Counter counter = new Counter(Counter.Mode.kExternalDirection);

@Override
public void robotInit() {
    // Set up the input channels for the counter
    counter.setUpSource(1);
```

(continues on next page)

(continued from previous page)

```

counter.setDownSource(2);

// Set the decoding type to 2X
counter.setUpSourceEdge(true, true);
}

```

C++

```

// Create a new Counter object in two-pulse mode
frc::Counter counter{frc::Counter::Mode::kExternalDirection};

void RobotInit() {
    // Set up the input channels for the counter
    counter.SetUpSource(1);
    counter.SetDownSource(2);

    // Set the decoding type to 2X
    counter.SetUpSourceEdge(true, true);
}

```

Configuring counter parameters

Note: The Counter class does not make any assumptions about units of distance; it will return values in whatever units were used to calculate the distance-per-pulse value. Users thus have complete control over the distance units used. However, units of time are *always* in seconds.

Note: The number of pulses used in the distance-per-pulse calculation does *not* depend on the decoding type - each “pulse” should always be considered to be a full cycle (rising and falling).

Apart from the mode-specific configurations, the Counter class offers a number of additional configuration methods:

Java

```

// Configures the counter to return a distance of 4 for every 256 pulses
// Also changes the units of getRate
counter.setDistancePerPulse(4./256.);

// Configures the counter to consider itself stopped after .1 seconds
counter.setMaxPeriod(.1);

// Configures the counter to consider itself stopped when its rate is below 10
counter.setMinRate(10);

// Reverses the direction of the counter
counter.setReverseDirection(true);

// Configures an counter to average its period measurement over 5 samples
// Can be between 1 and 127 samples
counter.setSamplesToAverage(5);

```

C++

```
// Configures the counter to return a distance of 4 for every 256 pulses
// Also changes the units of getRate
counter.SetDistancePerPulse(4./256.);

// Configures the counter to consider itself stopped after .1 seconds
counter.SetMaxPeriod(.1);

// Configures the counter to consider itself stopped when its rate is below 10
counter.SetMinRate(10);

// Reverses the direction of the counter
counter.SetReverseDirection(true);

// Configures an counter to average its period measurement over 5 samples
// Can be between 1 and 127 samples
counter.SetSamplesToAverage(5);
```

5.5.2 Reading information from counters

Regardless of mode, there is some information that the Counter class always exposes to users:

Count

Users can obtain the current count with the `get()` method:

Java

```
// returns the current count
counter.get();
```

C++

```
// returns the current count
counter.Get();
```

Distance

Note: Counters measure *relative* distance, not absolute; the distance value returned will depend on the position of the encoder when the robot was turned on or the encoder value was last *reset*.

If the *distance per pulse* has been configured, users can obtain the total distance traveled by the counted sensor with the `getDistance()` method:

Java

```
// returns the current distance
counter.getDistance();
```

C++

```
// returns the current distance
counter.GetDistance();
```

Rate

Note: Units of time for the Counter class are *always* in seconds.

Users can obtain the current rate of change of the counter with the `getRate()` method:

Java

```
// Gets the current rate of the counter
counter.getRate();
```

C++

```
// Gets the current rate of the counter
counter.GetRate();
```

Stopped

Users can obtain whether the counter is stationary with the `getStopped()` method:

Java

```
// Gets whether the counter is stopped
counter.getStopped();
```

C++

```
// Gets whether the counter is stopped
counter.GetStopped();
```

Direction

Users can obtain the direction in which the counter last moved with the `getDirection()` method:

Java

```
// Gets the last direction in which the counter moved
counter.getDirection();
```

C++

```
// Gets the last direction in which the counter moved
counter.GetDirection();
```

Period

Note: In *semi-period mode*, this method returns the duration of the pulse, not of the period.

Users can obtain the duration (in seconds) of the most-recent period with the `getPeriod()` method:

Java

```
// returns the current period in seconds
counter.getPeriod();
```

C++

```
// returns the current period in seconds
counter.GetPeriod();
```

5.5.3 Resetting a counter

To reset a counter to a distance reading of zero, call the `reset()` method. This is useful for ensuring that the measured distance corresponds to the actual desired physical measurement.

Java

```
// Resets the encoder to read a distance of zero
counter.reset();
```

C++

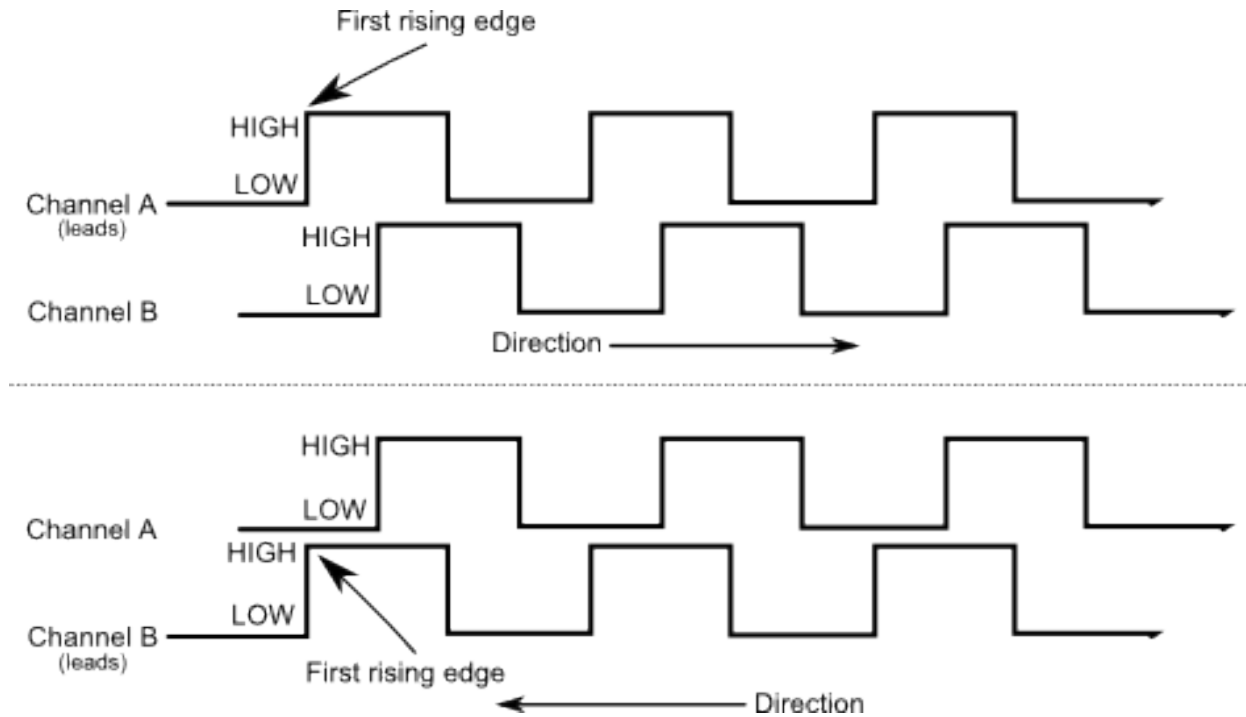
```
// Resets the encoder to read a distance of zero
counter.Reset();
```

5.5.4 Using counters in code

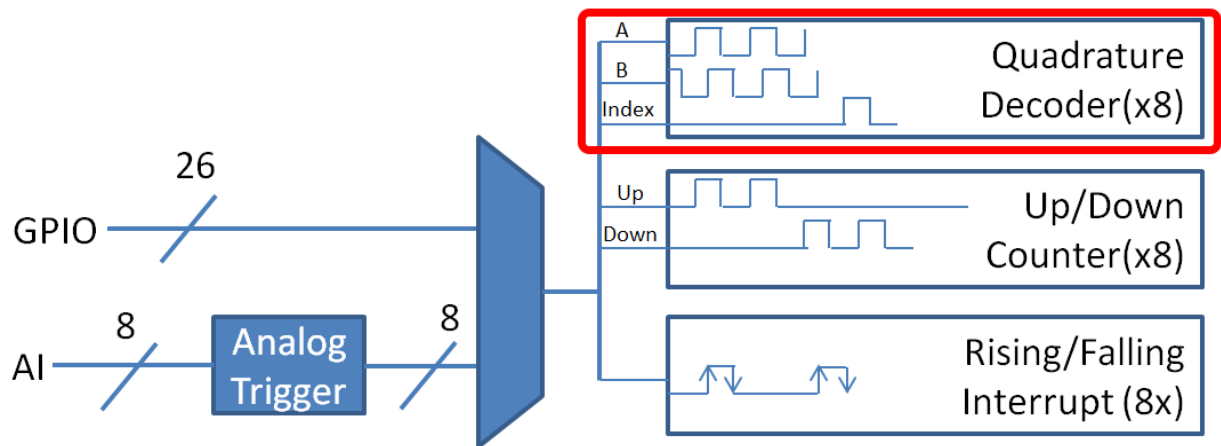
Counters are useful for a wide variety of robot applications - but since the `Counter` class is so varied, it is difficult to provide a good summary of them here. Many of these applications overlap with the `Encoder` class - a simple counter is often a cheaper alternative to a quadrature encoder. For a summary of potential uses for encoders in code, see [Encoders - Software](#).

5.6 Encoders - Software

Note: This section covers encoders in software. For a hardware guide to encoders, see [docs/hardware/sensors/encoders-hardware:Encoders - Hardware](#).



Encoders are devices used to measure motion (usually, the rotation of a shaft). The encoders used in FRC are known as “quadrature encoders.” These encoders produce square-wave signals on two channels that are a quarter-period out-of-phase (hence the term, “quadrature”). The pulses are used to measure the rotation, and the direction of motion can be determined from which channel “leads” the other.



The FPGA handles encoders either through a counter module or an encoder module, depending on the *decoding type* - the choice is handled automatically by WPILib. The FPGA contains 8 encoder modules.

5.6.1 The Encoder class

WPILib provides support for encoders through the Encoder class ([Java](#), [C++](#)). This class provides a simple API for configuring and reading data from encoders.

Initializing an encoder

An encoder can be instantiated as follows:

Java

```
// Initializes an encoder on DIO pins 0 and 1
// Defaults to 4X decoding and non-inverted
Encoder encoder = new Encoder(0, 1);
```

C++

```
// Initializes an encoder on DIO pins 0 and 1
// Defaults to 4X decoding and non-inverted
frc::Encoder encoder{0, 1};
```

Decoding type

The WPILib Encoder class can decode encoder signals in three different modes:

- **1X Decoding:** Increments the distance for every complete period of the encoder signal (once per four edges).
- **2X Decoding:** Increments the distance for every half-period of the encoder signal (once per two edges).
- **4X Decoding:** Increments the distance for every edge of the encoder signal (four times per period).

4X decoding offers the greatest precision, but at the potential cost of increased “jitter” in rate measurements. To use a different decoding type, use the following constructor:

Java

```
// Initializes an encoder on DIO pins 0 and 1
// 2X encoding and non-inverted
Encoder encoder = new Encoder(0, 1, false, Encoder.EncodingType.k2X);
```

C++

```
// Initializes an encoder on DIO pins 0 and 1
// 2X encoding and non-inverted
frc::Encoder encoder{0, 1, false, frc::Encoder::EncodingType::k2X};
```

Configuring encoder parameters

Note: The Encoder class does not make any assumptions about units of distance; it will return values in whatever units were used to calculate the distance-per-pulse value. Users thus have complete control over the distance units used. However, units of time are *always* in seconds.

Note: The number of pulses used in the distance-per-pulse calculation does *not* depend on the *decoding type* - each “pulse” should always be considered to be a full cycle (four edges).

The Encoder class offers a number of configuration methods:

Java

```
// Configures the encoder to return a distance of 4 for every 256 pulses
// Also changes the units of getRate
encoder.setDistancePerPulse(4./256.);

// Configures the encoder to consider itself stopped after .1 seconds
encoder.setMaxPeriod(.1);

// Configures the encoder to consider itself stopped when its rate is below 10
encoder.setMinRate(10);

// Reverses the direction of the encoder
encoder.setReverseDirection(true);

// Configures an encoder to average its period measurement over 5 samples
// Can be between 1 and 127 samples
encoder.setSamplesToAverage(5);
```

C++

```
// Configures the encoder to return a distance of 4 for every 256 pulses
// Also changes the units of getRate
encoder.SetDistancePerPulse(4./256.);

// Configures the encoder to consider itself stopped after .1 seconds
encoder.SetMaxPeriod(.1);

// Configures the encoder to consider itself stopped when its rate is below 10
encoder.SetMinRate(10);

// Reverses the direction of the encoder
encoder.SetReverseDirection(true);

// Configures an encoder to average its period measurement over 5 samples
// Can be between 1 and 127 samples
encoder.SetSamplesToAverage(5);
```

Reading information from encoders

The Encoder class provides a wealth of information to the user about the motion of the encoder.

Distance

Note: Quadrature encoders measure *relative* distance, not absolute; the distance value returned will depend on the position of the encoder when the robot was turned on or the

encoder value was last *reset*.

Users can obtain the total distance traveled by the encoder with the `getDistance()` method:

Java

```
// Configures an encoder to return a distance of 4 for every 256 pulses
encoder.setDistancePerPulse(4./256.);
```

C++

```
// Configures an encoder to return a distance of 4 for every 256 pulses
encoder.SetDistancePerPulse(4./256.);
```

Rate

Note: Units of time for the Encoder class are *always* in seconds.

Users can obtain the current rate of change of the encoder with the `getRate()` method:

Java

```
// Gets the current rate of the encoder
encoder.getRate();
```

C++

```
// Gets the current rate of the encoder
encoder.GetRate();
```

Stopped

Users can obtain whether the encoder is stationary with the `getStopped()` method:

Java

```
// Gets whether the encoder is stopped
encoder.getStopped();
```

C++

```
// Gets whether the encoder is stopped
encoder.GetStopped();
```

Direction

Users can obtain the direction in which the encoder last moved with the `getDirection()` method:

Java

```
// Gets the last direction in which the encoder moved
encoder.getDirection();
```

C++

```
// Gets the last direction in which the encoder moved
encoder.GetDirection();
```

Period

Users can obtain the period of the encoder pulses (in seconds) with the `getPeriod()` method:

Java

```
// Gets the current period of the encoder
encoder.getPeriod();
```

C++

```
// Gets the current period of the encoder
encoder.GetPeriod();
```

Resetting an encoder

To reset an encoder to a distance reading of zero, call the `reset()` method. This is useful for ensuring that the measured distance corresponds to the actual desired physical measurement, and is often called during a *homing* routine:

Java

```
// Resets the encoder to read a distance of zero
encoder.reset();
```

C++

```
// Resets the encoder to read a distance of zero
encoder.Reset();
```

5.6.2 Using encoders in code

Encoders are some of the most useful sensors in FRC; they are very nearly a requirement to make a robot capable of nontrivially-automated actuations and movement. The potential applications of encoders in robot code are too numerous to summarize fully here, but a few basic examples are provided below:

Driving to a distance

Encoders can be used on a robot drive to create a simple “drive to distance” routine. This is very useful for robot autonomy:

Java

```
// Creates an encoder on DIO ports 0 and 1
Encoder encoder = new Encoder(0, 1);

// Initialize motor controllers and drive
Spark left1 new Spark(0);
Spark left2 = new Spark(1);

Spark right1 = new Spark(2);
Spark right2 = new Spark(3);

SpeedControllerGroup leftMotors = new SpeedControllerGroup(left1, left2);
SpeedControllerGroup rightMotors = new SpeedControllerGroup(right1, right2);

DifferentialDrive drive = new DifferentialDrive(leftMotors, rightMotors);

@Override
public void robotInit() {
    // Configures the encoder's distance-per-pulse
    // The robot moves forward 1 foot per encoder rotation
    // There are 256 pulses per encoder rotation
    encoder.setDistancePerPulse(1./256.);
}

@Override
public void autonomousPeriodic() {
    // Drives forward at half speed until the robot has moved 5 feet, then stops:
    if(encoder.getDistance < 5) {
        drive.tankDrive(.5, .5);
    } else {
        drive.tankDrive(0, 0);
    }
}
```

C++

```
// Creates an encoder on DIO ports 0 and 1.
frc::Encoder encoder{0, 1};

// Initialize motor controllers and drive
frc::Spark left1{0};
frc::Spark left2{1};
frc::Spark right1{2};
frc::Spark right2{3};

frc::SpeedControllerGroup leftMotors{left1, left2};
frc::SpeedControllerGroup rightMotors{right1, right2};

frc::DifferentialDrive drive{leftMotors, rightMotors};

void Robot::RobotInit() {
    // Configures the encoder's distance-per-pulse
    // The robot moves forward 1 foot per encoder rotation
    // There are 256 pulses per encoder rotation
    encoder.SetDistancePerPulse(1./256.);
}

void Robot::AutonomousPeriodic() {
```

(continues on next page)

(continued from previous page)

```
// Drives forward at half speed until the robot has moved 5 feet, then stops:
if(encoder.GetDistance < 5) {
    drive.TankDrive(.5, .5);
} else {
    drive.TankDrive(0, 0);
}
}
```

Stabilizing heading

Warning: Like with all control loops, users should be careful to ensure that the sensor direction and the turning direction are consistent. If they are not, the loop will be unstable and the robot will turn wildly.

Encoders can be used to ensure that a robot drives straight in a manner quite similar to *how it is done with a gyroscope*. A simple implementation with a P loop is given below:

Java

```
// The encoders for the drive
Encoder leftEncoder = new Encoder(0,1);
Encoder rightEncoder = new Encoder(2,3);

// The gain for a simple P loop
double kP = 1;

// Initialize motor controllers and drive
Spark left1 = new Spark(0);
Spark left2 = new Spark(1);

Spark right1 = new Spark(2);
Spark right2 = new Spark(3);

SpeedControllerGroup leftMotors = new SpeedControllerGroup(left1, left2);
SpeedControllerGroup rightMotors = new SpeedControllerGroup(right1, right2);

DifferentialDrive drive = new DifferentialDrive(leftMotors, rightMotors);

@Override
public void autonomousInit() {
    // Configures the encoders' distance-per-pulse
    // The robot moves forward 1 foot per encoder rotation
    // There are 256 pulses per encoder rotation
    leftEncoder.setDistancePerPulse(1./256.);
    rightEncoder.setDistancePerPulse(1./256.);
}

@Override
public void autonomousPeriodic() {
    // Assuming no wheel slip, the difference in encoder distances is proportional to
    // the heading error
    double error = leftEncoder.getDistance() - rightEncoder.getDistance();
}
```

(continues on next page)

(continued from previous page)

```

    // Drives forward continuously at half speed, using the encoders to stabilize the
    ↪ heading
    drive.tankDrive(.5 + kP * error, .5 - kP * error);
}

```

C++

```

// The encoders for the drive
frc::Encoder leftEncoder{0,1};
frc::Encoder rightEncoder{2,3};

// The gain for a simple P loop
double kP = 1;

// Initialize motor controllers and drive
frc::Spark left1{0};
frc::Spark left2{1};
frc::Spark right1{2};
frc::Spark right2{3};

frc::SpeedControllerGroup leftMotors{left1, left2};
frc::SpeedControllerGroup rightMotors{right1, right2};

frc::DifferentialDrive drive{leftMotors, rightMotors};

void Robot::AutonomousInit() {
    // Configures the encoders' distance-per-pulse
    // The robot moves forward 1 foot per encoder rotation
    // There are 256 pulses per encoder rotation
    leftEncoder.SetDistancePerPulse(1./256.);
    rightEncoder.SetDistancePerPulse(1./256.);
}

void Robot::AutonomousPeriodic() {
    // Assuming no wheel slip, the difference in encoder distances is proportional to
    ↪ the heading error
    double error = leftEncoder.GetDistance() - rightEncoder.GetDistance();

    // Drives forward continuously at half speed, using the encoders to stabilize the
    ↪ heading
    drive.TankDrive(.5 + kP * error, .5 - kP * error);
}

```

More-advanced implementations can use more-complicated control loops. Closing a control loop on the encoder difference is roughly analogous to closing it on the heading error, and so PD loops are particularly effective.

PID Control

Encoders are particularly useful as inputs to PID controllers (the heading stabilization example above is a simple P loop).

Homing an encoded mechanism

Since encoders measure *relative* distance, it is often important to ensure that their “zero-point” is in the right place. A typical way to do this is a “homing routine,” in which a mechanism is moved until it hits a known position (usually accomplished with a limit switch), or “home,” and then the encoder is reset. The following code provides a basic example:

Java

```
Encoder encoder = new Encoder(0, 1);

Spark spark = new Spark(0);

// Limit switch on DIO 2
DigitalInput limit = new DigitalInput(2);

public void autonomousPeriodic() {
    // Runs the motor backwards at half speed until the limit switch is pressed
    // then turn off the motor and reset the encoder
    if(!limit.get()) {
        spark.set(-.5);
    } else {
        spark.set(0);
        encoder.reset();
    }
}
```

C++

```
frc::Encoder encoder{0,1};

frc::Spark spark{0};

// Limit switch on DIO 2
frc::DigitalInput limit{2};

void AutonomousPeriodic() {
    // Runs the motor backwards at half speed until the limit switch is pressed
    // then turn off the motor and reset the encoder
    if(!limit.Get()) {
        spark.Set(-.5);
    } else {
        spark.Set(0);
        encoder.Reset();
    }
}
```

5.7 Entradas analógicas - Software

Note: Esta seção cobre entradas analógicas no software. Para obter um guia de hardware para entradas analógicas, consulte [docs/hardware/sensors/analog-inputs-hardware:Analog Inputs - Hardware](#).

O FPGA do roboRIO suporta até 8 canais de entrada analógica que podem ser usados para ler o valor de uma tensão analógica de um sensor. As entradas analógicas podem ser usadas para qualquer sensor que produz uma tensão simples.

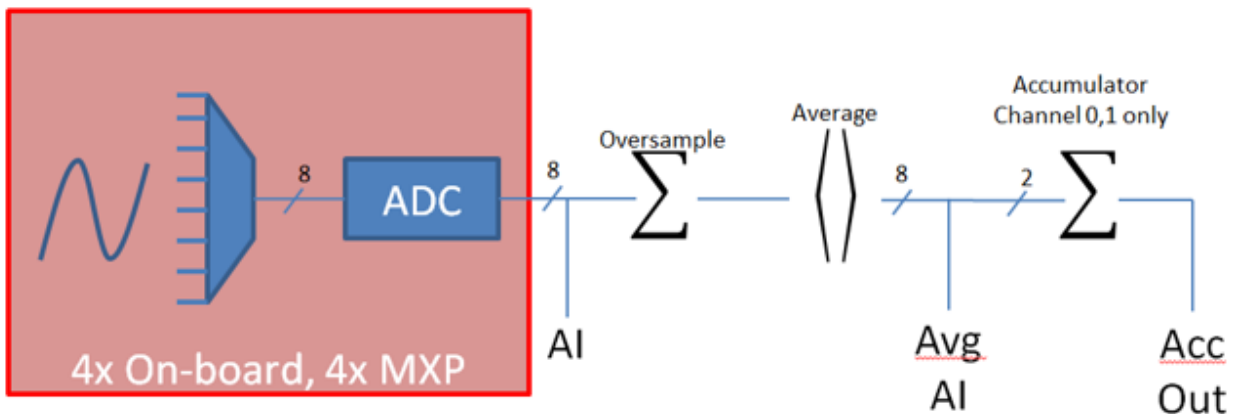
As entradas analógicas do FPGA, por padrão, retornam um número inteiro de 12 bits proporcional à tensão, de 0 a 5 volts.

5.7.1 Entradas analógicas - explicação

Note: Geralmente, é mais conveniente usar a explicação de invólucro de *Analog Potentiometers* do que usar AnalogInput diretamente, pois suporta o dimensionamento para unidades significativas.

O suporte para leitura das tensões nas entradas analógicas do FPGA é fornecido através da AnalogInput explicação (Java, C++).

Sobre-amostragem e Média



Os módulos de entrada analógica do FPGA suportam tanto a sobre amostragem quanto a média. Esses comportamentos são altamente semelhantes, mas diferem em alguns aspectos importantes. Ambos podem ser usados ao mesmo tempo.

Sobre-amostragem

Quando a superamostragem está ativada, o FPGA adiciona várias amostras consecutivas e retorna o valor acumulado. Os usuários podem especificar o número de bits de oversampling - para nbits de oversampling, o número de amostras somadas é $2 \sim \{n\}$.

Média

A média se comporta de maneira semelhante à superamostragem, exceto que os valores acumulados são divididos pelo número de amostras, para que a escala dos valores retornados não seja alterada. Isso geralmente é mais conveniente, mas ocasionalmente o erro adicional de arredondamento introduzido pelo arredondamento é indesejável.

Note: Quando a sobre amostragem e a média são usadas ao mesmo tempo, a sobre amostragem é aplicada primeiro e , em seguida, os valores sobre amostragem são calculados. Portanto, a superamostragem de 2 bits e a média de 2 bits usadas ao mesmo tempo aumentam a escala dos valores retornados em aproximadamente um fator de 2 e diminuem a taxa de atualização em aproximadamente um fator de 4.

Lendo valores de uma entrada analógica

Os valores podem ser lidos em um AnalogInput com um dos quatro métodos diferentes:

Obter valor

O `getValue` método retorna o valor medido instantâneo bruto da entrada analógica, sem aplicar nenhuma calibração e ignorar as configurações de superamostragem e média. O valor retornado é um número inteiro.

Obter voltagem

O `getVoltage` método retorna a tensão instantânea medida da entrada analógica. As configurações de superamostragem e média são ignoradas, mas o valor é redimensionado para representar uma tensão. O valor retornado é um dobro.

Obter valor médio

O `getAverageValue` método retorna o valor médio da entrada analógica. O valor não é redimensionado, mas a superamostragem e a média são aplicadas. O valor retornado é um número inteiro.

Obter voltagem média

O `getAverageVoltage` método retorna a tensão média da entrada analógica. Reescalamento, superamostragem e média são todos aplicados. O valor retornado é um dobro.

Acumulador

Note: Atualmente, os métodos do acumulador não suportam o retorno de um valor em unidades de volts - o valor retornado sempre será um número inteiro (especificamente, a `long`).

Os canais de entrada analógica 0 e 1 suportam adicionalmente um acumulador, que integra (soma) o sinal indefinidamente, de modo que o valor retornado seja a soma de todos os valores medidos passados. Sobre-amostragem e média são aplicadas antes da acumulação.

Obtendo contagem e valor sincronizados

Às vezes, é necessariamente obter medições correspondentes da contagem e do valor. Isso pode ser feito usando o `getAccumulatorOutput` método.

5.7.2 Usando entradas analógicas no código

A `AnalogInput` explicação pode ser usada para escrever código para uma ampla variedade de sensores (incluindo potenciômetros, acelerômetros, giroscópios, ultrassônicos e outros) que retornam seus dados como uma tensão analógica. No entanto, se possível, é quase sempre mais conveniente usar uma das outras classes WPIlib existentes que manipula o código de nível inferior (lendo as tensões analógicas e convertendo-as em unidades significativas) para você. Os usuários devem usar diretamente apenas `AnalogInput` como “último recurso”.

Assim, para exemplos de como usar efetivamente sensores analógicos no código, os usuários devem consultar as outras páginas deste capítulo que tratam de explicações mais específicas.

5.8 Potenciômetros analógicos - Software

Note: Esta seção aborda potenciômetros analógicos em software. Para obter um guia de hardware para potenciômetros analógicos, consulte `docs/hardware/sensors/analog-potentiometers-hardware:Analog Potentiometers - Hardware`.

Potenciômetros são resistores variáveis que permitem converter informações sobre a posição em um sinal de tensão analógico. Esse sinal pode ser lido pelo roboRIO para controlar qualquer dispositivo conectado ao potenciômetro.

Embora seja possível ler informações de um potenciômetro diretamente com um [Entradas analógicas - Software](#), o WPIlib fornece uma `AnalogPotentiometer` explicação (Java, C++) que lida com a redimensionamento dos valores em unidades significativas para o usuário. É altamente recomendável usar esta explicação.

De fato, o `AnalogPotentiometer` nome é um nome impróprio - essa classe deve ser usada para a grande maioria dos sensores que retornam seu sinal como uma tensão analógica simples, em escala linear.

5.8.1 Explicação dos Potenciômetros analógicos

Note: Os parâmetros “faixa completa” ou “escala” no `AnalogPotentiometer` construtor são fatores de escala de um intervalo de 0-1 ao intervalo real, não de 0-5. Ou seja, eles representam uma escala fracionária nativa, em vez de uma escala de tensão.

Personalizando o Potenciômetro analógico subjacente

Note: Se o usuário alterar a escala da AnalogInput com superamostragem, isso deve ser refletido na configuração de escala passada para o

5.8.2 Usando potenciômetros analógicos no código

Os sensores analógicos podem ser usados no código da mesma maneira que outros sensores que medem a mesma coisa. Se o sensor analógico for um potenciômetro medindo um ângulo do braço, ele poderá ser usado de maneira semelhante a um *encoder*. Se for um sensor ultrassônico, pode ser usado de maneira semelhante a outros *ultrasonics*.

É muito importante ter em mente que os potenciômetros físicos reais geralmente têm uma amplitude de movimento limitada. As salvaguardas devem estar presentes tanto no mecanismo físico quanto no código, para garantir que o mecanismo não quebre o sensor passando seu lance máximo.

5.9 Digital Inputs - Software

Note: This section covers digital inputs in software. For a hardware guide to digital inputs, see docs/hardware/sensors/digital-inputs-hardware:Digital Inputs - Hardware.

The roboRIO's FPGA supports up to 26 digital inputs. 10 of these are made available through the built-in DIO ports on the RIO itself, while the other 16 are available through the MXP breakout port.

Digital inputs read one of two states - "high" or "low." By default, the built-in ports on the RIO will read "high" due to internal pull-up resistors (for more information, see docs/hardware/sensors/digital-inputs-hardware:Digital Inputs - Hardware). Accordingly, digital inputs are most-commonly used with switches of some sort. Support for this usage is provided through the DigitalInput class (Java, C++).

5.9.1 The DigitalInput class

A DigitalInput can be initialized as follows:

Java

```
// Initializes a DigitalInput on DIO 0
DigitalInput input = new DigitalInput(0);
```

C++

```
// Initializes a DigitalInput on DIO 0
frc::DigitalInput input{0};
```

Reading the value of the DigitalInput

The state of the DigitalInput can be polled with the get method:

Java

```
// Gets the value of the digital input. Returns true if the circuit is open.
input.get();
```

C++

```
// Gets the value of the digital input. Returns true if the circuit is open.
input.Get();
```

5.9.2 Creating a DigitalInput from an AnalogInput

Note: An AnalogTrigger constructed with a port number argument can share that analog port with a separate AnalogInput, but two *AnalogInput* objects may not share the same port.

Sometimes, it is desirable to use an analog input as a digital input. This can be easily achieved using the AnalogTrigger class (Java, C++).

An AnalogTrigger may be initialized as follows. As with AnalogPotentiometer, an AnalogInput may be passed explicitly if the user wishes to customize the sampling settings:

Java

```
// Initializes an AnalogTrigger on port 0
AnalogTrigger trigger0 = new AnalogTrigger(0);

// Initializes an AnalogInput on port 1 and enables 2-bit oversampling
AnalogInput input = new AnalogInput(1);
input.setAverageBits(2);

// Initializes an AnalogTrigger using the above input
AnalogTrigger trigger1 = new AnalogTrigger(input);
```

C++

```
// Initializes an AnalogTrigger on port 0
frc::AnalogTrigger trigger0{0};

// Initializes an AnalogInput on port 1 and enables 2-bit oversampling
frc::AnalogInput input{1};
input.SetAverageBits(2);

// Initializes an AnalogTrigger using the above input
frc::AnalogTrigger trigger1{input};
```

Setting the trigger points

Note: For details on the scaling of “raw” AnalogInput values, see *Entradas analógicas - Software*.

To convert the analog signal to a digital one, it is necessary to specify at what values the trigger will enable and disable. These values may be different to avoid “dithering” around the transition point:

Java

```
// Sets the trigger to enable at a raw value of 3500, and disable at a value of 1000
trigger.setLimitsRaw(1000, 3500);

// Sets the trigger to enable at a voltage of 4 volts, and disable at a value of 1.5
↪volts
trigger.setLimitsVoltage(1.5, 4);
```

C++

```
// Sets the trigger to enable at a raw value of 3500, and disable at a value of 1000
trigger.SetLimitsRaw(1000, 3500);

// Sets the trigger to enable at a voltage of 4 volts, and disable at a value of 1.5
↪volts
trigger.SetLimitsVoltage(1.5, 4);
```

5.9.3 Using DigitalInputs in code

As almost all switches on the robot will be used through a DigitalInput, this class is extremely important for effective robot control.

Limiting the motion of a mechanism

Nearly all motorized mechanisms (such as arms and elevators) in FRC should be given some form of “limit switch” to prevent them from damaging themselves at the end of their range of motions. A short example is given below:

Java

```
Spark spark = new Spark(0);

// Limit switch on DIO 2
DigitalInput limit = new DigitalInput(2);

public void autonomousPeriodic() {
    // Runs the motor forwards at half speed, unless the limit is pressed
    if(!limit.get()) {
        spark.set(.5);
    } else {
        spark.set(0);
    }
}
```

C++

```
// Motor for the mechanism
frc::Spark spark{0};

// Limit switch on DIO 2
frc::DigitalInput limit{2};

void AutonomousPeriodic() {
    // Runs the motor forwards at half speed, unless the limit is pressed
    if(!limit.Get()) {
        spark.Set(.5);
    } else {
        spark.Set(0);
    }
}
```

Homing an encoded mechanism

Limit switches are very important for being able to “home” an encoded mechanism. For an example of this, see [Homing an encoded mechanism](#).

6.1 Using CAN Devices

CAN has many advantages over other methods of connection between the robot controller and peripheral devices.

- CAN connections are daisy-chained from device to device, which often results in much shorter wire runs than having to wire each device to the RIO itself.
- Much more data can be sent over a CAN connection than over a PWM connection - thus, CAN motor controllers are capable of a much more expansive feature-set than are PWM motor controllers.
- CAN is bi-directional, so CAN motor controllers can send data back to the RIO, again facilitating a more expansive feature-set than can be offered by PWM Controllers.

Supported CAN peripherals include:

- CAN speed controllers
- The Power Distribution Panel (PDP)
- The Pneumatics Control Module (PCM)

For instructions on wiring CAN devices, see the relevant section of the robot wiring guide.

CAN devices generally have their own WPILib classes. The following sections will describe the use of several of these classes.

6.2 Pneumatics Control Module

The Pneumatics Control Module (PCM) is a CAN-based device that provides complete control over the compressor and up to 8 solenoids per module. The PCM is integrated into WPILib through a series of classes that make it simple to use.

The closed loop control of the Compressor and Pressure switch is handled by the Compressor class (Java, C++), and the Solenoids are handled by the Solenoid (Java, C++) and DoubleSolenoid (Java, C++) classes.

An additional PCM module can be used where the modules corresponding solenoids are differentiated by the module number in the constructors of the Solenoid and Compressor classes.

For more information on controlling the compressor, see [Operating a Compressor for Pneumatics](#).

For more information on controlling solenoids, see [Operating Pneumatic Cylinders](#).

6.3 Power Distribution Panel

The Power Distribution Panel (PDP) can use its CAN connectivity to communicate a wealth of status information regarding the robot's power use to the roboRIO, for use in user code. The PDP has the capability to report its current temperature, the bus voltage, the total robot current draw, the total robot energy use, and the individual current draw of each device power channel. These data can be used for a number of advanced control techniques, such as motor torque limiting and brownout avoidance.

6.3.1 Creating a PDP Object

To use the PDP, create an instance of the `PowerDistributionPanel` class ([Java](#), [C++](#)):

C++

```
PowerDistributionPanel examplePDP{0};
```

Java

```
PowerDistributionPanel examplePDP = new PowerDistributionPanel(0);
```

Note: it is not necessary to create a `PowerDistributionPanel` object unless you need to read values from it. The board will work and supply power on all the channels even if the object is never created.

Warning: To enable voltage and current logging in the Driver Station, the CAN ID for the PDP *must* be 0.

6.3.2 Reading the Bus Voltage

Java

```
examplePDP.getVoltage();
```

C++

```
examplePDP.GetVoltage();
```

Monitoring the bus voltage can be useful for (among other things) detecting when the robot is near a brownout, so that action can be taken to avoid brownout in a controlled manner.

6.3.3 Reading the Temperature

Java

```
examplePDP.getTemperature();
```

C++

```
examplePDP.GetTemperature();
```

Monitoring the temperature can be useful for detecting if the robot has been drawing too much power and needs to be shut down for a while, or if there is a short or other wiring problem.

6.3.4 Reading the Total Current and Energy

Java

```
examplePDP.getTotalCurrent();  
examplePDP.getTotalEnergy();
```

C++

```
examplePDP.GetTotalCurrent();  
examplePDP.GetTotalEnergy();
```

Monitoring the total current and total energy (the total energy is simply the total current multiplied by the bus voltage) can be useful for controlling how much power is being drawn from the battery, both for preventing brownouts and ensuring that mechanisms have sufficient power available to perform the actions required.

6.3.5 Reading Individual Channel Currents

The PDP also allows users to monitor the current drawn by the individual device power channels. For example, to read the current on channel 0:

Java

```
examplePDP.getCurrent(0);
```

C++

```
examplePDP.GetCurrent(0);
```

Monitoring individual device current draws can be useful for detecting shorts or stalled motors.

6.4 Third-Party CAN Devices

A number of FRC vendors offer their own CAN peripherals. As CAN devices offer expansive feature-sets, vendor CAN devices require similarly expansive code libraries to operate. As a result, these libraries are not maintained as an official part of WPILib, but are instead

maintained by the vendors themselves. For a guide to installing third-party libraries, see [3rd Party Libraries](#)

A list of common third-party CAN devices from various vendors, along with links to corresponding external documentation, is provided below:

6.4.1 Cross-the-Road Electronics

Cross-the-Road Electronics (CTRE) offers several CAN peripherals with external libraries:

CTRE Motor Controllers

- **Talon SRX**
 - [API Documentation \(Java, C++\)](#)
 - [Technical Manual](#)
- **Victor SPX**
 - [API Documentation \(Java, C++\)](#)
 - [Technical Manual](#)

CTRE Sensors

- **Pigeon IMU**
 - [API Documentation \(Java, C++\)](#)
 - [Technical Manual](#)
- **CANifier**
 - [API Documentation \(Java, C++\)](#)
 - [Technical Manual](#)

6.4.2 REV Robotics

REV Robotics currently offers the SPARK MAX motor controller, which has a similar feature-set to the Talon SRX.

REV Motor Controllers

- **SPARK MAX**
 - [API Documentation \(Java, C++\)](#)
 - [Technical Manual](#)

6.4.3 Playing With Fusion

Playing With Fusion (PWF) offers the Venom integrated motor/controller as well as a Time-of-Flight distance sensor:

PWF Motor Controllers

- **Venom**
 - [API Documentation \(Java, C++\)](#)
 - [Technical Manual](#)

PWF Sensors

- **Time of Flight Sensor**
 - [API Documentation\(Java, C++\)](#)
 - [Technical Manual](#)

6.5 FRC CAN Device Specifications

This document seeks to describe the basic functions of the current FRC CAN system and the requirements for any new CAN devices seeking to work with the system.

6.5.1 Addressing

FRC CAN nodes assign arbitration IDs based on a pre-defined scheme that breaks the ID into 5 components:

Device Type

This is a 5-bit value describing the type of device being addressed. A table of currently assigned device types can be found below. If you wish to have a new device type assigned from the Reserved pool, please submit a request to FIRST.

Device Types	
Broadcast Messages	0
Robot Controller	1
Motor Controller	2
Relay Controller	3
Gyro Sensor	4
Accelerometer	5
Ultrasonic Sensor	6
Gear Tooth Sensor	7
Power Distribution Module	8
Pneumatics Controller	9
Miscellaneous	10
IO Breakout	11
Reserved	12-30
Firmware Update	31

Manufacturer

This is an 8-bit value indicating the manufacturer of the CAN device. Currently assigned values can be found in the table below. If you wish to have a manufacturer ID assigned from the Reservedpool, please submit a request to FIRST.

Manufacturer	
Broadcast	0
NI	1
Luminary Micro	2
DEKA	3
CTR Electronics	4
REV Robotics	5
Grapple	6
MindSensors	7
Team Use	8
Kauai Labs	9
Copperforge	10
Playing With Fusion	11
Studica	12
Reserved	13-255

API/Message Identifier

The API or Message Identifier is a 10-bit value that identifies a particular command or message type. These identifiers are unique for each Manufacturer + Device Type combination (so an API identifier that may be a “Voltage Set” for a Luminary Micro Motor Controller may be a “Status Get” for a CTR Electronics Motor Controller or Current Get for a CTR Power Distribution Module).

The Message identifier is further broken down into 2 sub-fields: the 6-bit API Class and the 4-bit API Index.

API Class

The API Class is a 6-bit identifier for an API grouping. Similar messages are grouped into a single API Class. An example of the API Classes for the Jaguar Motor Controller is shown in the table below.

API Class	
Voltage Control Mode	0
Speed Control Mode	1
Voltage Compensation Mode	2
Position Control Mode	3
Current Control Mode	4
Status	5
Periodic Status	6
Configuration	7
Ack	8

API Index

The API Index is a 4-bit identifier for a particular message within an API Class. An example of the API Index values for the Jaguar Motor Controller Speed Control API Class is shown in the table below.

API Index	
Enable Control	0
Disable Control	1
Set Setpoint	2
P Constant	3
I Constant	4
D Constant	5
Set Reference	6
Trusted Enable	7
Trusted Set No Ack	8
Trusted Set Setpoint No Ack	10
Set Setpoint No Ack	11

Device Number

Device Number is a 6-bit quantity indicating the number of the device of a particular type. Devices should default to device ID 0 to match other components of the FRC Control System. Device 0x3F may be reserved for device specific broadcast messages.

Example

Speed Control Mode Disable from Luminary Micro Jaguar Speed Controller (dev # 4)

	Device Type					Manufacturer Code								API						Device Number									
Field Value	2					2								API Class 1						Index 1		4							
Bits	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0		
Bit Position	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

6.5.2 Protected Frames

FRC CAN Nodes which implement actuator control capability (motor controllers, relays, pneumatics controllers, etc.) must implement a way to verify that the robot is enabled and that commands originate with the main robot controller (i.e. the roboRIO).

6.5.3 Broadcast Messages

Broadcast messages are messages sent to all nodes by setting the device type and manufacturer fields to 0. The API Class for broadcast messages is 0. The currently defined broadcast messages are shown in the table below:

Description	
Disable	0
System Halt	1
System Reset	2
Device Assign	3
Device Query	4
Heartbeat	5
Sync	6
Update	7
Firmware Version	8
Enumerate	9
System Resume	10

Devices should disable immediately when receiving the Disable message (arbID 0), implementation of other broadcast messages is optional.

6.5.4 Requirements for FRC CAN Nodes

For CAN Nodes to be accepted for use in the FRC System, they must:

- Communicate using Arbitration IDs which match the prescribed FRC format:
 - A valid, issued CAN Device Type (per Table 1 - CAN Device Types)
 - A valid, issued Manufacturer ID (per Table 2 - CAN Manufacturer Codes)
 - API Class(es) and Index(s) assigned and documented by the device manufacturer

- A user selectable device number if multiple units of the device type are intended to co-exist on the same network.
- Support the minimum Broadcast message requirements as detailed in the Broadcast Messages section.
- If controlling actuators, utilize a scheme to assure that the robot is issuing commands, is enabled, and is still present
- Provide software library support for LabVIEW, C++, and Java or arrange with FIRST or FIRSTs Control System Partners to provide such interfaces.

7.1 Git Version Control Introduction

Important: A more in-depth guide on Git is available on the [Git website](#).

[Git](#) is a Distributed Version Control System (VCS) created by Linus Torvalds, also known for creating and maintaining the linux kernel. Version Control is a system for tracking changes of code for developers. The advantages of Git Version Control are:

- Separate testing environments into *branches*
- Ability to navigate to a particular *commit* without removing history
- Ability to manage *commits* in various ways, including combining them
- Various other features, see [here](#)

7.1.1 Prerequisites

Important: This tutorial uses the Windows operating system

You have to download and install Git from the following links:

- [Windows](#)
- [macOS](#)
- [Linux](#)

Note: You may need to add Git to your [path](#)

7.1.2 Git Vocabulary

Git revolves around several core commands:

- **Repository:** the data structure of your code, including a `.git` folder in the root directory
- **Commit:** a particular saved state of the repository, this includes all files and additions
- **Branch:** a means of separating various commits, having a unique history. This is primarily used for separating development and stable branches.
- **Push:** update the remote repository with your local changes
- **Pull:** update your local repository with the remote changes
- **Clone:** retrieving a local copy of a repository to modify
- **Fork:** duplicating a pre-existing repository to modify, and to compare against the original
- **Merge:** combining various changes from different branches/commits/forks into a single history

7.1.3 Repository

A Git repository is a data structure containing the structure, history, and files of a project.

Git repositories usually consist of:

- A `.git` folder. This folder contains the various information about the repository.
- A `.gitignore` file. This file contains the files or directories that you do *not* want included when you commit.
- Files and folders. This is the main content of the repository.

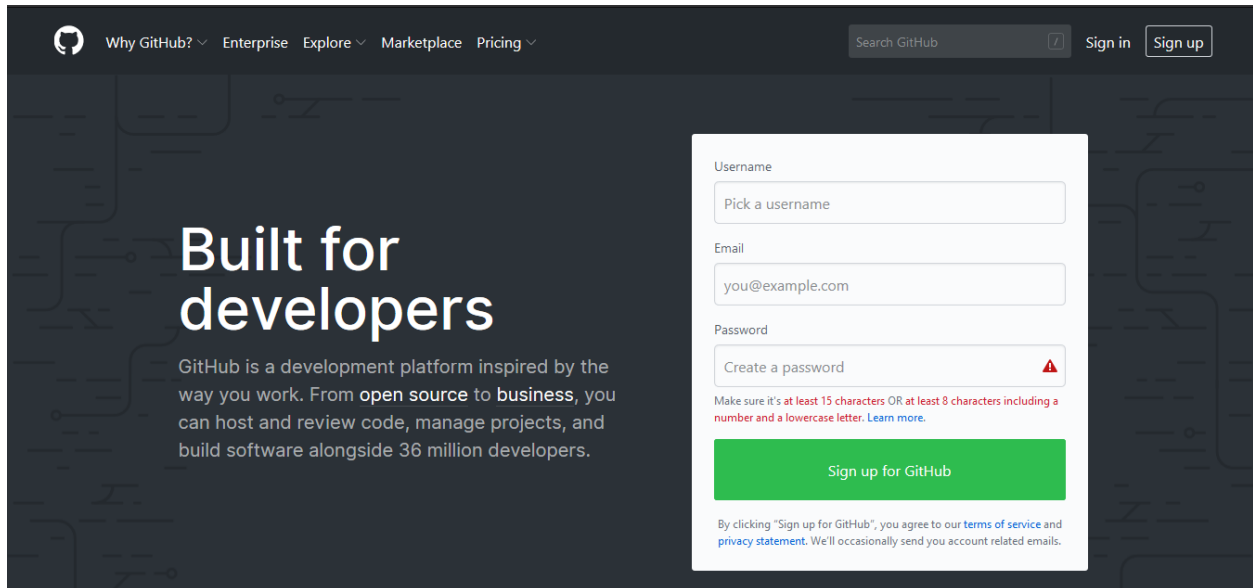
Creating the repository

You can store the repository locally, or through a remote. A remote being the cloud, or possibly another storage medium that hosts your repository. [Github](#) is a popular free hosting service. Numerous developers use it, and that's what this tutorial will use.

Note: There are various providers that can host repositories. [Gitlab](#), [Bitbucket](#), and [Cloud-forge](#) are a few alternatives to Github

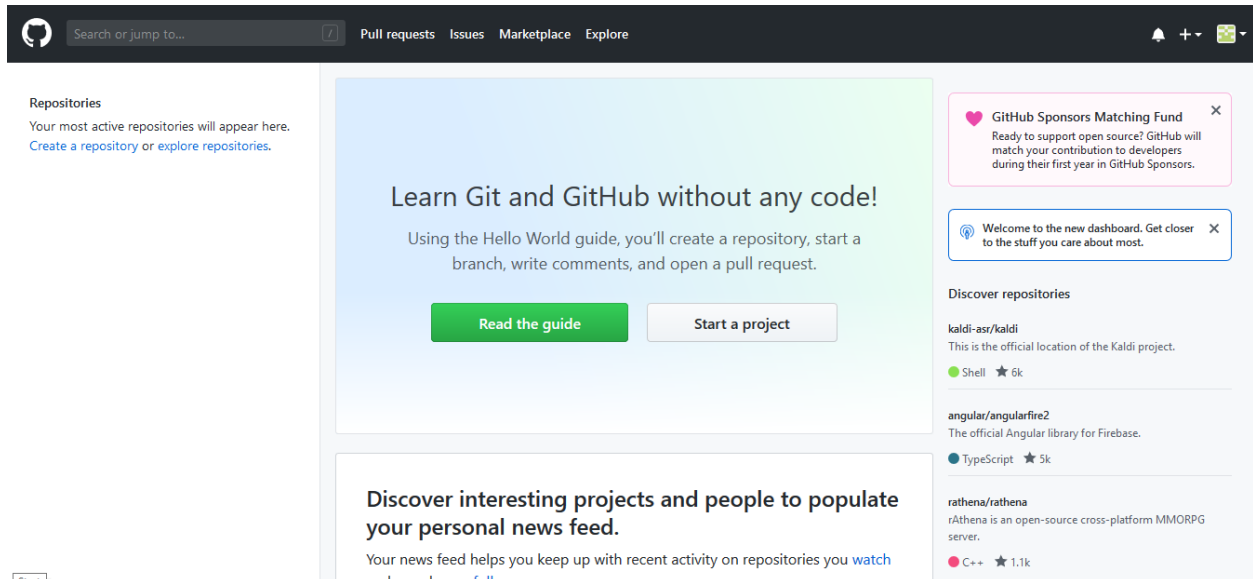
Creating a Github Account

Go ahead and create a Github account by visiting the [website](#) and following the own screen prompts.

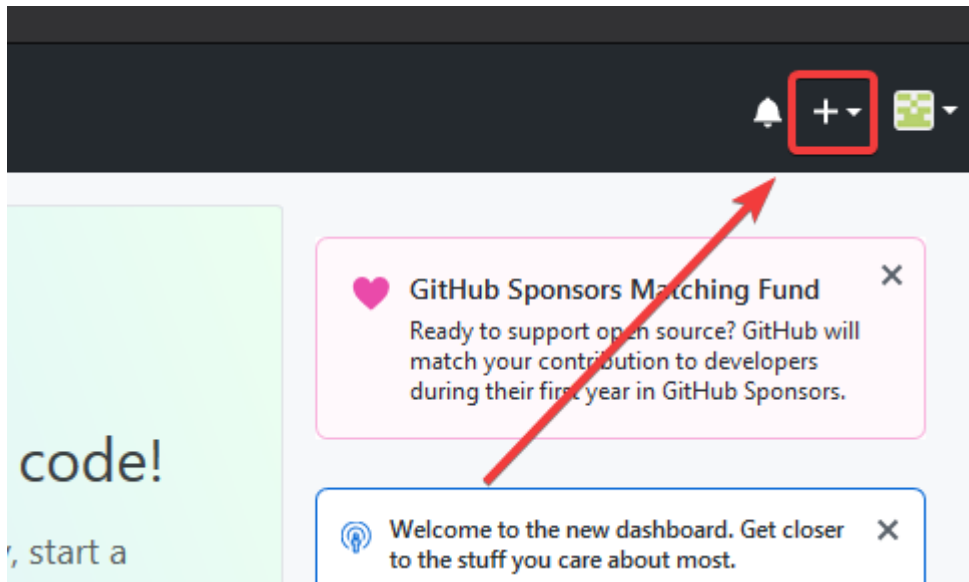


Local Creation

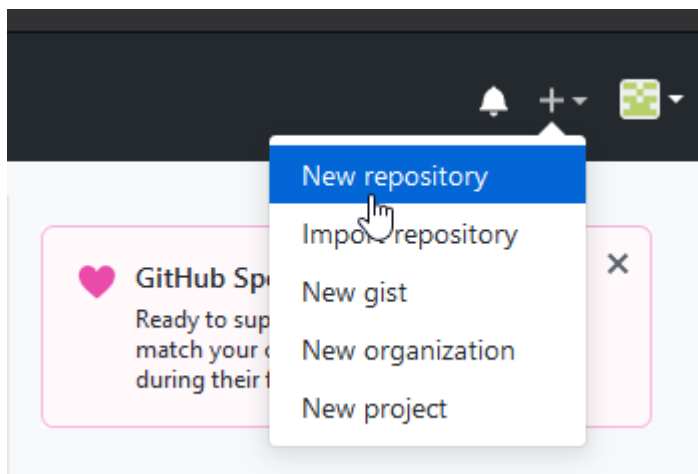
After creating and verifying your account, you'll want to visit the homepage. It'll look similar to the shown image.



Click the plus icon in the top right.



Then click “New Repository”




Fill out the appropriate information, and then click “Create repository”

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Owner

Repository name *

 ExampleUser9007 ▾ / ExampleRepo ✓

Great repository names are short and memorable. Need inspiration? How about [reimagined-palm-tree?](#)

Description (optional)



Public

Anyone can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

☐ Initialize this repository with a README

This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore: None ▾



Add a license: None ▾



Create repository

You should see a screen similar to this


Quick setup — if you've done this kind of thing before

 Set up in Desktop or **HTTPS** **SSH** 

Get started by [creating a new file](#) or [uploading an existing file](#). We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).


...or create a new repository on the command line

```
echo "# ExampleRepo" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/ExampleUser9007/ExampleRepo.git
git push -u origin master
```




...or push an existing repository from the command line

```
git remote add origin https://github.com/ExampleUser9007/ExampleRepo.git
git push -u origin master
```



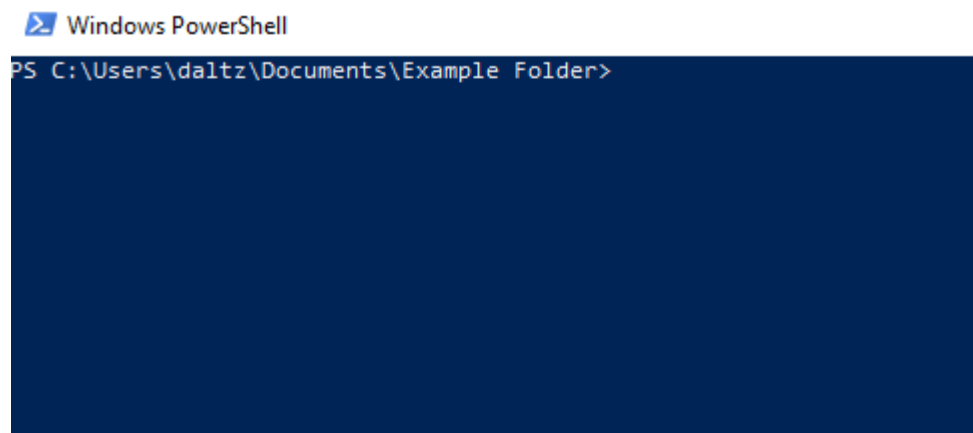
...or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

 Import code

Note: The keyboard shortcut `ctrl + ~` can be used to open a terminal in Visual Studio Code.

Now you'll want to open a powershell window and navigate to your project directory. An excellent tutorial on powershell can be found [here](#). Please consult your search engine on how to open a terminal on alternative operating systems.



In the below example, we created a file called `README.md` with the contents of `# Example Repo`. More details on the various commands can be found in the subsequent sections.

```
> cd "C:\Users\ExampleUser9007\Documents\Example Folder"
> git init
```

(continues on next page)

(continued from previous page)

```
Initialized empty Git repository in C:/Users/ExampleUser9007/Documents/Example Folder/
↪.git/
> echo "# ExampleRepo" >> README.md
> git add README.md
> git commit -m "First commit"
[master (root-commit) fafafa] First commit
 1 file changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 README.md
> git remote add origin https://github.com/ExampleUser9007/ExampleRepo.git
> git push -u origin master
```

7.1.4 Commits

Repositories are primarily composed of commits. Commits are saved states or *versions* of code.

In the previous example, we created a file called README.md. Open that file in your favorite text editor and edit a few lines. After tinkering with the file for a bit, simply save and close. Navigate to powershell and type the following commands.

```
> git add README.md
> git commit -m "Adds a description to the repository"
[master bcbcbc] Adds a description to the repository
 1 file changed, 2 insertions(+), 0 deletions(-)
> git push
```

Note: Writing good commit messages is a key part of a maintainable project. A guide on writing commit messages can be found [here](#).

Git Pull

Note: `git fetch` can be used when the user does not wish to automatically merge into the current working branch

This command retrieves the history or commits from the remote repository. When the remote contains work you do not have, it will attempt to automatically merge. See [Merging](#).

Run: `git pull`

Git Add

This command adds a selected file(s) to a commit. To commit every file/folder that isn't excluded via *gitignore*.

Run: `git add FILENAME.txt` where FILENAME.txt is the name and extension of the file to add to a commit. Run: `git add .` will add every untracked, unexcluded file when ran in the root of the repository.

Git Commit

This command creates the commit and stores it locally. This saves the state and adds it to the repositories history.

Run: `git commit -m "type message here"`

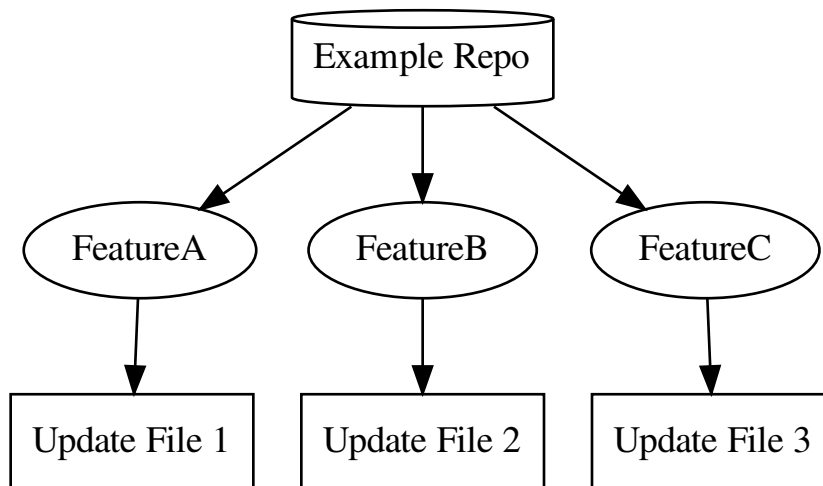
Git Push

Upload (Push) your local changes to the remote (Cloud)

Run: `git push`

7.1.5 Branches

Branches are a similar to parallel worlds to Git. They start off the same, and then they can “branch” out into different varying paths. Consider the Git control flow to look similar to this.



In the above example, FeatureB was merged into FeatureA. This is what is called a merge. You are “merging” the changes from one branch into another.

Creating a Branch

Run: `git branch branch-name` where branch-name is the name of the branch to create. The new branch history will be created from the current active branch.

Entering a Branch

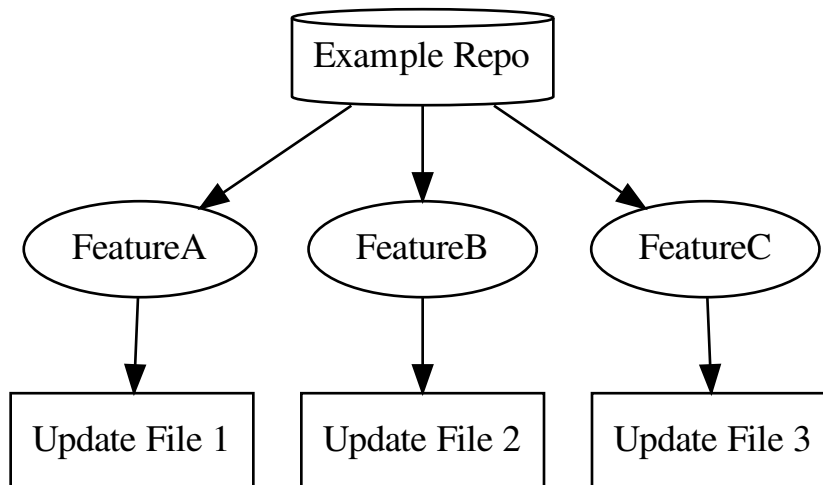
Once a branch is created, you have to then enter the branch.

Run: `git checkout branch-name` where `branch-name` is the branch that was previously created.

7.1.6 Merging

In scenarios where you want to copy one branches history into another, you can merge them. A merge is done by calling `git merge branch-name` with `branch-name` being the name of the branch to merge from. It is automatically merged in the current active branch.

It's common for a remote repository to contain work (history) that you do not have. Whenever you run `git pull`, it will attempt to automatically merge those commits. That merge may look like the below.



However, in the above example, what if File 1 was modified by both branch FeatureA and FeatureB? This is called a **merge conflict**. A merge conflict will can be resolved by editing the conflicting file. In the example, we would need to edit File 1 to keep the history or changes that we want. After that has been done. Simply re-add, re-commit, and then push your changes.

7.1.7 Resets

Sometimes history needs to be reset, or a commit needs to be undone. This can be done multiple ways.

Reverting the Commit

Note: You cannot revert a merge, as git does not know which branch or origin it should choose.

To revert history leading up to a commit run `git revert commit-id`. The commit IDs can be shown using the `git log` command.

Resetting the Head

Warning: Forcibly resetting the head is a dangerous command. It permanently erases all history past the target. You have been warned!

Run: `git reset --hard commit-id`.

7.1.8 Forks

Forks can be treated similarly to branches. You can merge the upstream (original repository) into the origin (forked repository).

Updating a Fork

1. Add the upstream: `git remote add upstream https://github.com/ORIGINAL_OWNER/ORIGINAL_REPOSITORY.git`
2. Confirm it was added via: `git remote -v`
3. Pull changes from upstream: `git fetch upstream`
4. Merge the changes into head: `git merge upstream/upstream-branch-name`

7.1.9 Gitignore

Important: It is extremely important that teams **do not** modify the `.gitignore` file that is included with their robot project. This can lead to offline deployment not working.

A `.gitignore` file is commonly used as a list of files to not automatically commit with `git add`. Any files or directory listed in this file will **not** be committed. They will also not show up with `git status`.

Additional Information can be found [here](#)

Hiding a Folder

Simply add a new line containing the folder to hide, with a forward slash at the end

EX: `directory-to-exclude/`

Hiding a File

Add a new line with the name of the file to hide, including any prepending directory relative to the root of the repository.

EX: `directory/file-to-hide.txt`

EX: `file-to-hide2.txt`

7.1.10 Additional Information

A much more in-depth tutorial can be found at the official [git](#) website.

A guide for correcting common mistakes can be found at the [git flight rules](#) repository.

Além da documentação aqui, há uma variedade de outros recursos disponíveis para os times de FRC® para ajudar no entendimento do Sistema de Controle e de Softwares.

8.1 Outras documentações

Além deste site, existem outros locais onde as equipes podem verificar a documentação:

- [NI FRC Community Documents Section](#)
- [FIRST Inspires Technical Resources Page](#)
- [CTRE Product Pages](#)

8.2 Fóruns

Tem uma pergunta não respondida pela documentação? O suporte é fornecido aqui:

- [NI FRC Community Discussion Section](#) (perguntas sobre roboRIO, LabVIEW e Driver Station software)
- [FIRST Inspires Control System Forum](#) (perguntas sobre wiring, hardware e Driver Station)
- [FIRST Inspires Programming Forum](#) (perguntas sobre programação com C++, Java, ou LabVIEW)

8.3 Under Control 1156

Alguma pergunta não respondida? Entre em contato com o time pelo nosso email: robotica.1156@gmail.com

8.4 Suporte via telefone da NI

Tem uma pergunta sobre LabVIEW, roboRIO, ou sobre a Driver Station? A NI oferece suporte via telefone para os times de FRC durante a build season em +1 (866) 511-6285 1:00-7:00 PM CST Segunda - Sexta.

8.5 CTRE

Suporte para os componentes da Cross The Road Electronics (Pneumatics Control Module, Power Distribution Panel, Talon SRX, e Voltage Regulator Module) é oferecido pelo email support@crosstheroadelectronics.com

8.6 Reporte de Bugs

Encontrou um bug? Informe-nos relatando-o na seção Issues do projeto WPILibSuite apropriado no Github: <https://github.com/wpilibsuite>

Phoenix Software Reference Manual

This is the latest documentation for CTR-Electronics Phoenix software framework. This includes:

- 1) Class library for Talon SRX, Victor SPX, CANifier and Pigeon IMU (C++/Java/LabVIEW for FRC, C# for HERO).
- 2) Phoenix Tuner GUI - provides configuration options, diagnostics, control and plotting.
- 3) Phoenix Diagnostic Server - install on to RIO for Tuner, and to perform HTTP API requests for diagnostic information.

Be sure to follow the following instructions in order to ensure success in developing your robot platform.

Primer: CTRE CAN Devices

CTR-Electronics has designed many of the available CAN bus devices for FRC-style robotics. This includes:

- Talon FX, Talon SRX, and Victor SPX motor controllers (PWM and CAN)
- CANCoder
- Pigeon IMU
- CANifier
- Pneumatics Control Module (PCM)
- Power Distribution Panel (PDP)

These devices have similar functional requirements, specifically every device of a given model group requires a unique device ID for typical FRC use (settings, control and status). The device ID is usually expressed as a number between '0' and '62', allowing use for up to 63 Talon SRXs, 63 Victors, 63 PDPs, etc. at once. This range does not intercept with device IDs of other CAN device types. For example, there is no harm in having a Pneumatics Control Module (PCM) and a Talon SRX both with device ID '0'. However, having two Talon SRXs with device ID '0' will be problematic.

These devices are field upgradable, and the firmware shipped with your devices will pre-date the "latest and greatest" tested firmware intended for use with the latest API release. Firmware update can be done easily using Phoenix Tuner.

The Talon FX/SRX and Victor SPX provide two pairs of twisted CANH (yellow) and CANL (green) allowing for daisy chaining. Other devices such as the PDP and PCM have Weidmuller connectors that accept twisted pair cabling. Often you will be able to use your Talons and Victors to connect together your PCM and PDP to each other.

The CAN termination resistors are built into the FRC robot controller (roboRIO) and in the Power Distribution Panel (PDP) assuming the PDP's termination jumper is in the ON position.

More information on wiring and hardware requirements can be found in the user manual of each device type.

Primer: What is Phoenix Software

Phoenix is a package that targets LabVIEW, C++, and Java for the FRC Robotics Controller platform, i.e. the NI roboRIO robot controller.

It includes the Application Programming Interface (API), which are the functions you call to manipulate the CTRE CAN bus devices: Talon FX, Talon SRX, Victor SPX, CANCoder, CANifier, and Pigeon IMU.

Note: PCM and PDP API are built into the core WPI distribution.

The C++ and Java APIs are very similar, typically only differing on the function name case (configAllSettings in Java versus ConfigAllSettings in C++). Because Java is more widely used in FRC than C++, this document will reference the Java routine names. C++ users should take note that the leading character of every function is UPPERCASE in C++.

Additionally, Phoenix shared libraries are also targeted for C++ on Linux (amd64, armhf, aarch64) and typically available on our maven repository. The example support libraries use socket-can for CANBus access, however custom drivers can be provided by the end user for alternative CANBus solutions (NVIDIA TX2 native CAN bus for example).

Phoenix also includes a NETMF (C#) class library for the non-FRC HERO Robot Controller. This can replace the roboRIO in use cases that don't require the full features of the FRC control system, and are not in use during competition.

Note: With Phoenix framework, teams can control/leverage Talons, Victors, Pigeons, CAN-Coders, CANifiers outside of the roboRIO (e.g. Rasp-Pi or Jetson TX2), and use the roboRIO/DriverStation to safely enable/disable the actuators.

Note: Leveraging CTRE CAN devices from third-party CAN hardware was officially made FRC legal for the **2019 season**.

There are tons of examples in all languages at CTRE's GitHub account:

- <https://github.com/CrossTheRoadElec/Phoenix-Examples-Languages>
- <https://github.com/CrossTheRoadElec/Phoenix-Examples-LabVIEW>

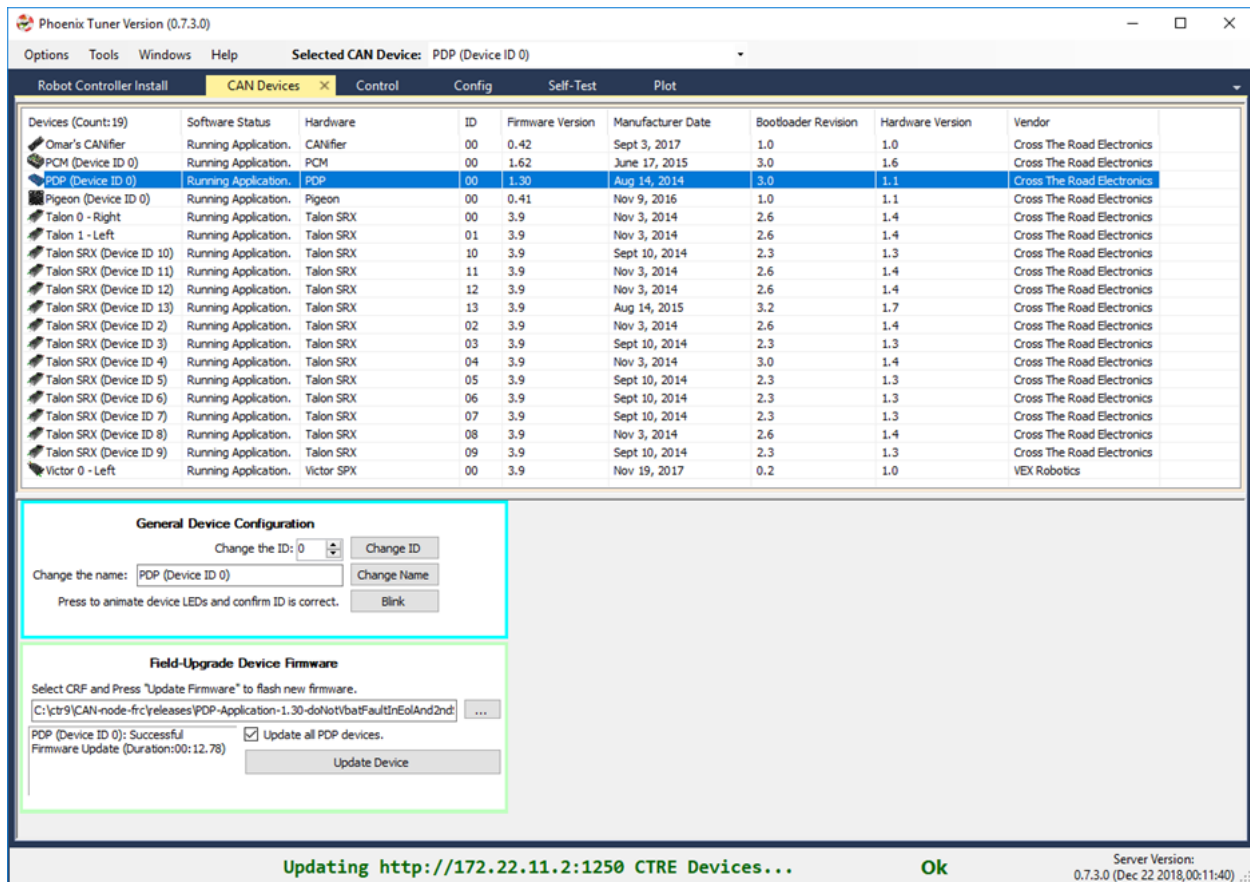
Entire GitHub organization: <https://github.com/CrossTheRoadElec/>

Phoenix-Examples-Languages and Phoenix-Examples-LabVIEW are specifically tested on the FRC RoboRIO control system.

Phoenix-Linux-SocketCAN-Example demonstrates control of Talons from a Raspberry Pi. <https://github.com/CrossTheRoadElec/Phoenix-Linux-SocketCAN-Example>

11.1 What is Phoenix Tuner?

Phoenix-Tuner is the graphical interface that allows for configuration of Phoenix CAN bus devices.



It provides a variety of functionality to support all Phoenix CAN Bus devices. The feature set includes:

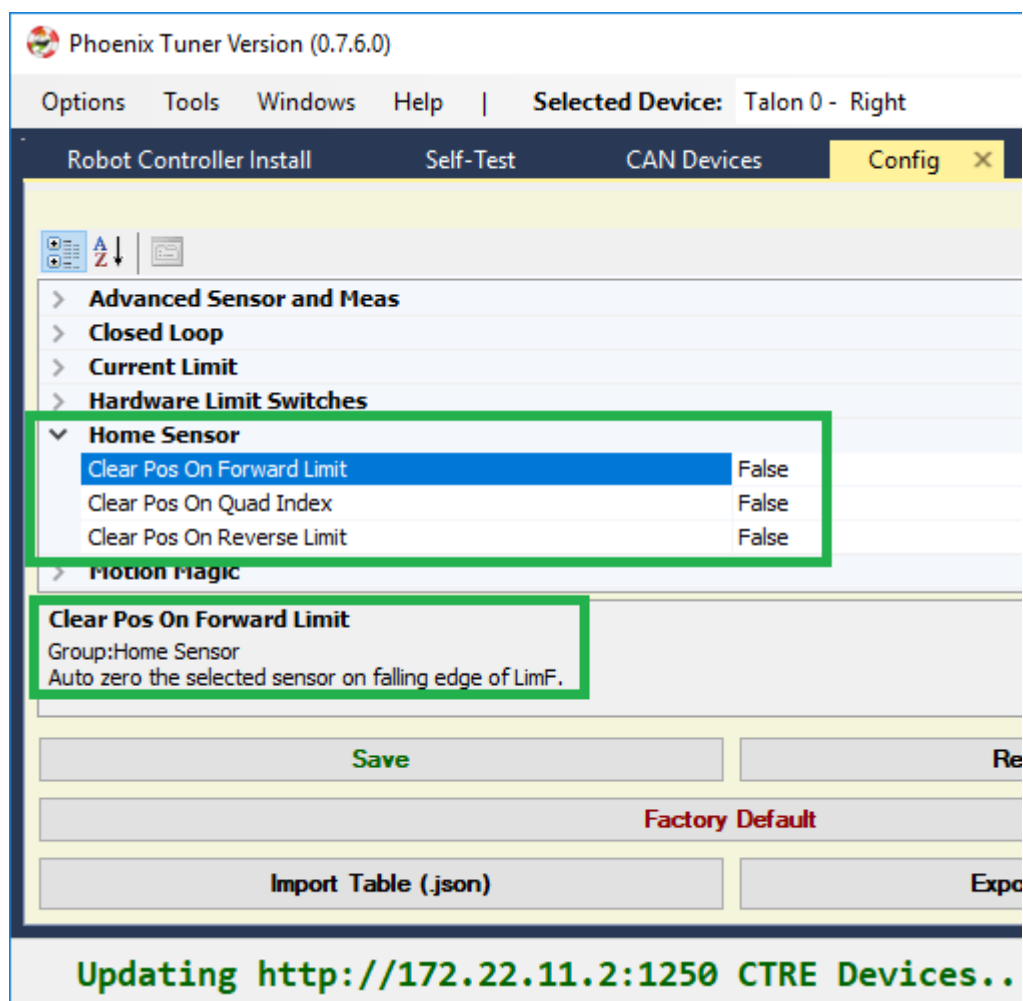
- Update device firmware (including PDP/PCM)
- Change CAN IDs
- Configure direction and offsets
- Self-test Snapshot devices
- Change configuration settings
- Factory default configuration settings
- Test motors

- Check plots
- Temperature Calibrate Pigeon-IMU
- Confirm proper CAN bus wiring **without writing any software.**

Now you can drive your motors and collect data *without writing any software.*



Configuration values can be **checked, modified, and defaulted** with the new config view. Config values can also be **imported/exported** as an easy-to-follow JSON formatted file.



The following sections of documentation will cover how to use Phoenix Tuner and the other components of Phoenix.

Tip: Have a feature request? Send to us at support@ctr-electronics.com or report it on GitHub.

Prepare your workstation computer

12.1 Before Installing Phoenix...

It is strongly recommended to complete the base installation of FRC tools. <https://docs.wpilib.org/en/latest/docs/getting-started/getting-started-frc-control-system/control-system-software.html>

Warning: You will need to image the roboRIO to 2020 software before continuing. The roboRIO kickoff versions are **image 2020_v10**.

12.1.1 Test base FRC Installation - FRC LabVIEW

If a team intends to use LabVIEW to develop robot software, be sure to complete the full NI installer. At which point, opening LabVIEW should reveal the FRC-styled graphical start menu.

At this point it is recommended to create a simple template project and test deploy to the roboRIO. Be sure the DriverStation can communicate with the robot controller, and that DS message log is functional.

Note: LabVIEW is versioned 2019 due to its release schedule. Therefore, LV2019 is used for the 2020 season.

12.1.2 Test base FRC Installation - FRC C++ / Java

It is recommended to install the FRC Driver Station Utilities. This will install the Driver Station software, which is necessary for:

1. Basic comms checks
2. Reading joystick data

3. Generally required for enabling motor actuation (Phoenix Tuner Control features may require this, depending on setup).

12.1.3 General Recommendations for FRC C++ / Java.

The FRC C++/Java standard distribution for 2020 is based on the Microsoft Visual Studio Code development environment with WPI extensions.

If you are not familiar with developing C++/Java FRC programs, we strongly recommend testing full deployment to your robot controller before installing Phoenix and porting previous season software. A recommended test is to:

1. Create a project from scratch
2. Make a simple change such as add a print statement with a counter.
3. Deploy (or debug) your application.
4. Confirm robot can be teleop-enabled in DS.
5. Open FRC message Console and read through all the messages.
6. Repeat 2 - 5 ten times. This will train students to become familiar with and build general confidence in the tools.

Note: Third-party vendor libraries are installed into the C++/Java project, not the environment. For each C++/Java project you create, you must use the WPI provided tools to select Phoenix to bring the libraries into your project.

12.2 What to Download (and why)

12.2.1 Option 1: Windows installer (strongly recommended)

Environments: Windows-LabVIEW, Windows-C++/Java, HERO C#

Phoenix Installer zip can be downloaded at:

http://www.ctr-electronics.com/hro.html#product_tabs_technical_resources.

It is typically named Phoenix Framework_Windows_vW.X.Y.Z.zip

This will install:

- The LabVIEW Phoenix API (if LabVIEW is detected and selected in installer)
- The C++/Java Phoenix API (if selected in installer)
- Device Firmware Files (that were tested with the release)
- CTRE Support of RobotBuilder
- Phoenix Tuner
 - Installs Phoenix API libraries into the roboRIO (required for LabVIEW)
 - Installs Phoenix Diagnostics Server into the RoboRIO (needed for CAN diagnostics).
 - Plotter/Control features

- Self-test Snapshot
- Device ID and field-upgrade

12.2.2 Option 2: Phoenix API via Non-Windows Zip

Environments: Linux/MacOS - C++/Java

The Phoenix API can be manually installed on non-Windows platforms by downloading the “non-Windows” zip and following the instructions found inside.

This essentially contains a maven-style repository that holds the API binaries and headers, as well as a “vendordeps” JSON file that instructs VS how to incorporate the Phoenix C++/Java API libraries.

Note: This is auto installed when using the Windows full installer (Option 1).

12.2.3 Phoenix Tuner

Environments: Windows

If you are using option 2, you will need to download Phoenix Tuner separately. Phoenix Tuner is available here... <https://github.com/CrossTheRoadElec/Phoenix-Tuner-Release/releases>

This can be convenient for workstations that aren’t used for software development, but are used for field-upgrade or testing motor controllers.

Note: LabVIEW teams may need to use Phoenix Tuner to install Phoenix libraries into the roboRIO. More information on this can be found under Prepare Robot Controller.

Note: This is auto installed when using the Windows full installer.

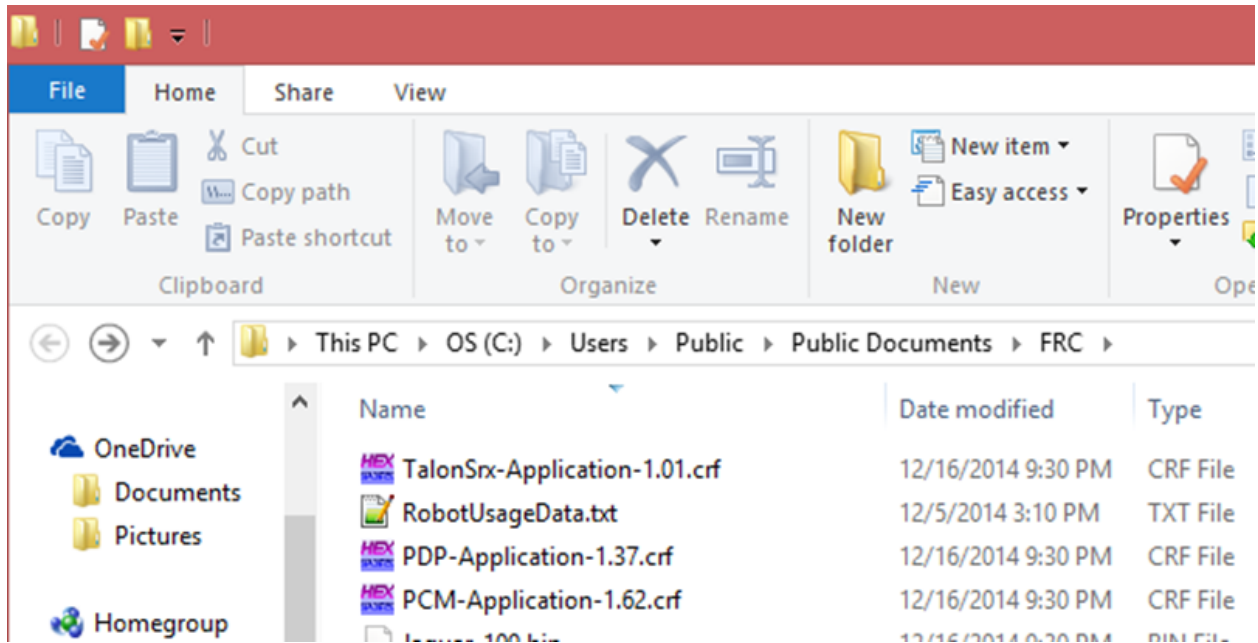
Note: Developers may be interested to know that all Phoenix Tuner features leverage an HTTP API provided by the Phoenix Diagnostics Server. As such, custom tooling can be developed to field-upgrade, test-control, or diagnostics CTRE devices without Tuner.

12.2.4 Device Firmware Files (crf)

The test firmware files for all CTRE devices are packaged with the Windows Installer (and has been for years). However, many FRC teams don’t notice, or prefer to download them directly from the product pages on the ctr-electronics.com website. If Internet access is available, they can be downloaded as such.

The FRC Software installer will create a directory with various firmware files/tools for many control system components. Typically, the path is:

C:\Users\Public\Documents\FRC



When the path is entered into a browser, the browser may fix-up the path:

C:\Users\Public\Public Documents\FRC

In this directory are the initial release firmware CRF files for all CTRE CAN bus devices, including the new Talon FX and CANCoder.

The latest firmware to be used can be found in the [ch22_SoftReleaseNote](#).

Note: Additionally, newer updates may be provided online at <http://www.ctr-electronics.com>.

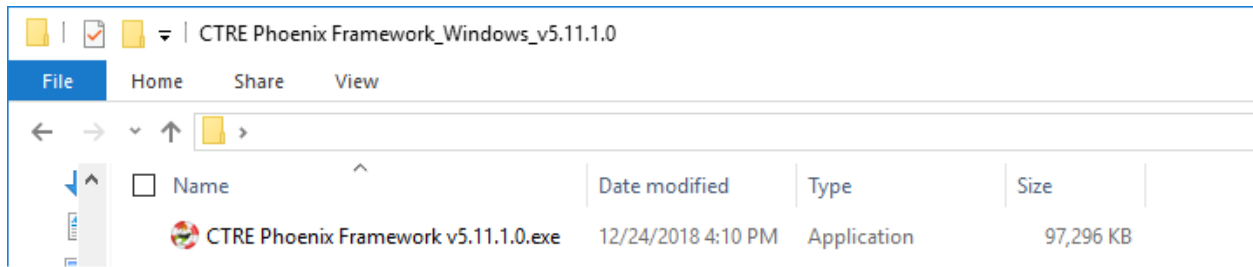
Note: There is no longer FRC versus non-FRC firmware for motor controllers. Instead the latest firmware detects if the use case is FRC. If so, the device will FRC-Lock, and will require the Driver Station for actuation.

12.3 Workstation Installation

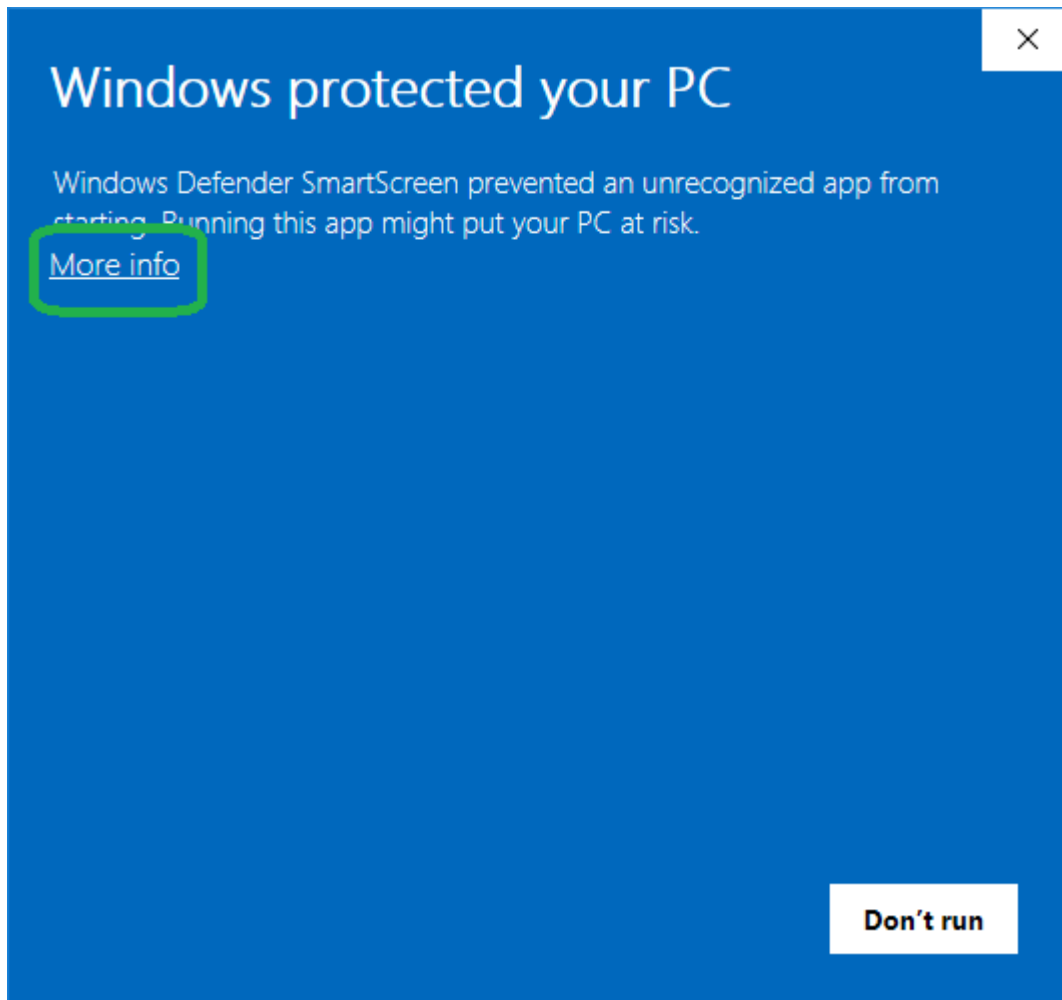
There are three installation methods listed below. The simplest and recommended approach is to run the Windows Installer (Option 1).

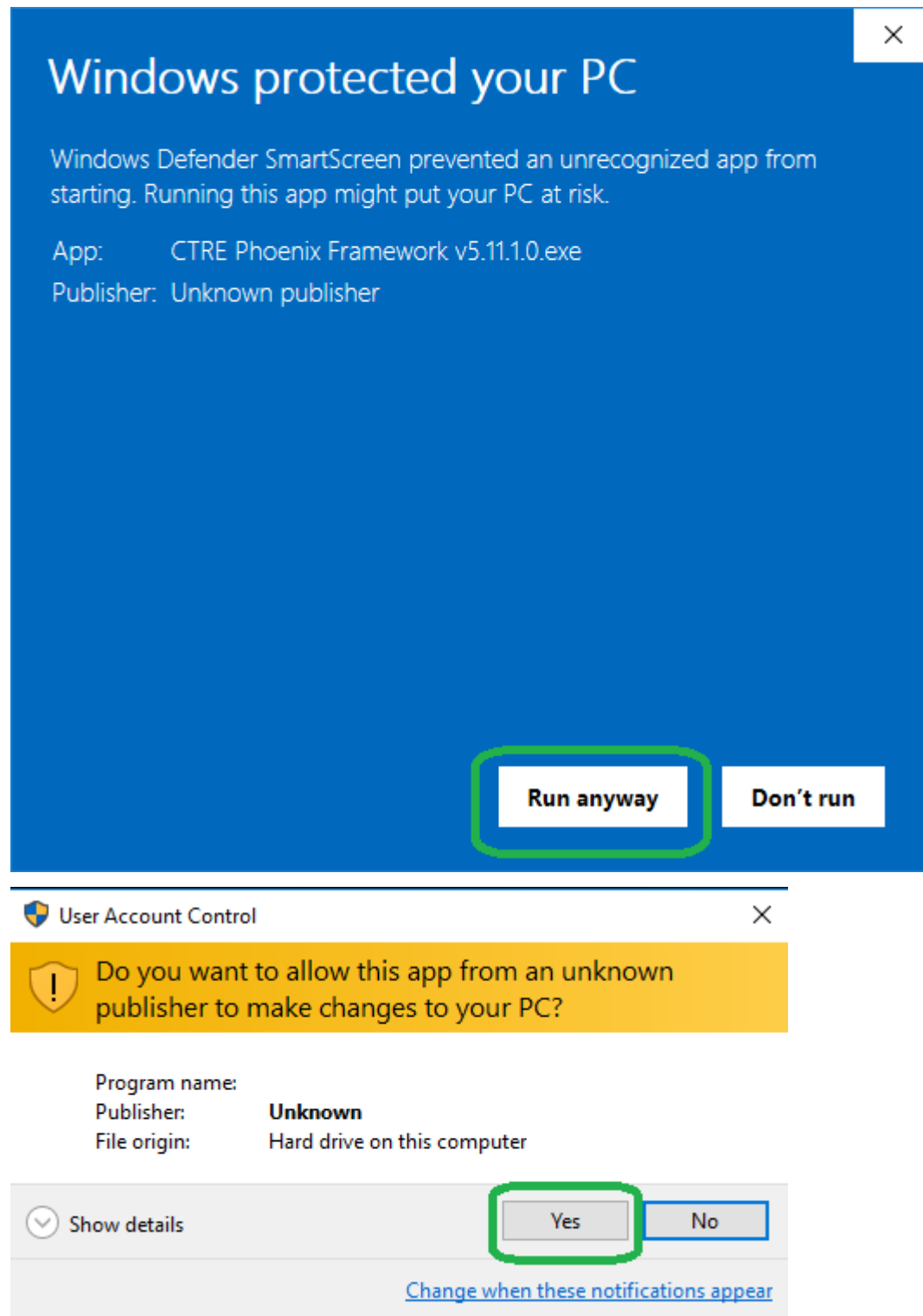
12.3.1 Option 1: Windows Offline Installer (C++/Java/LabVIEW, HERO C#)

Un-compress the downloaded zip.

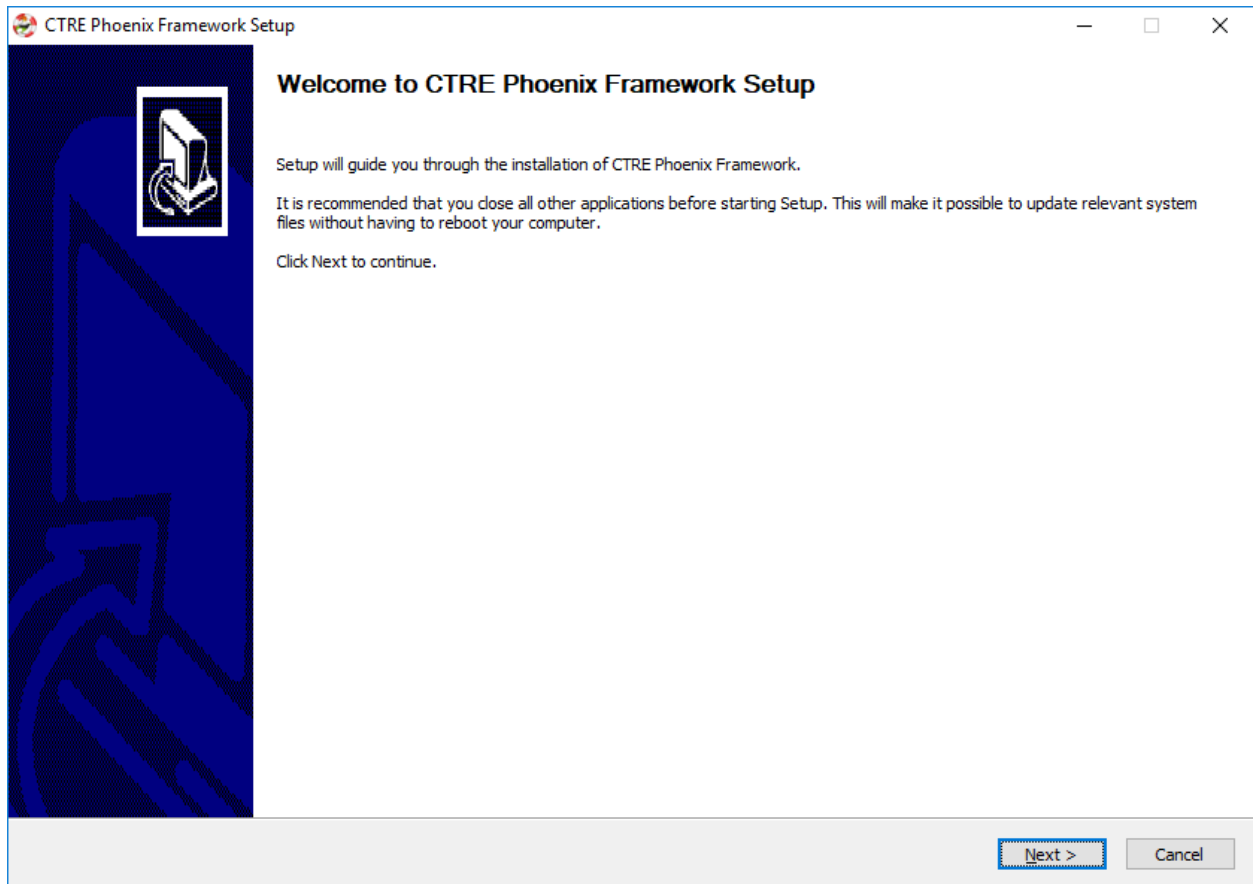


Double click on the installer. If the Windows protection popup appears press More Info, then Run anyway.





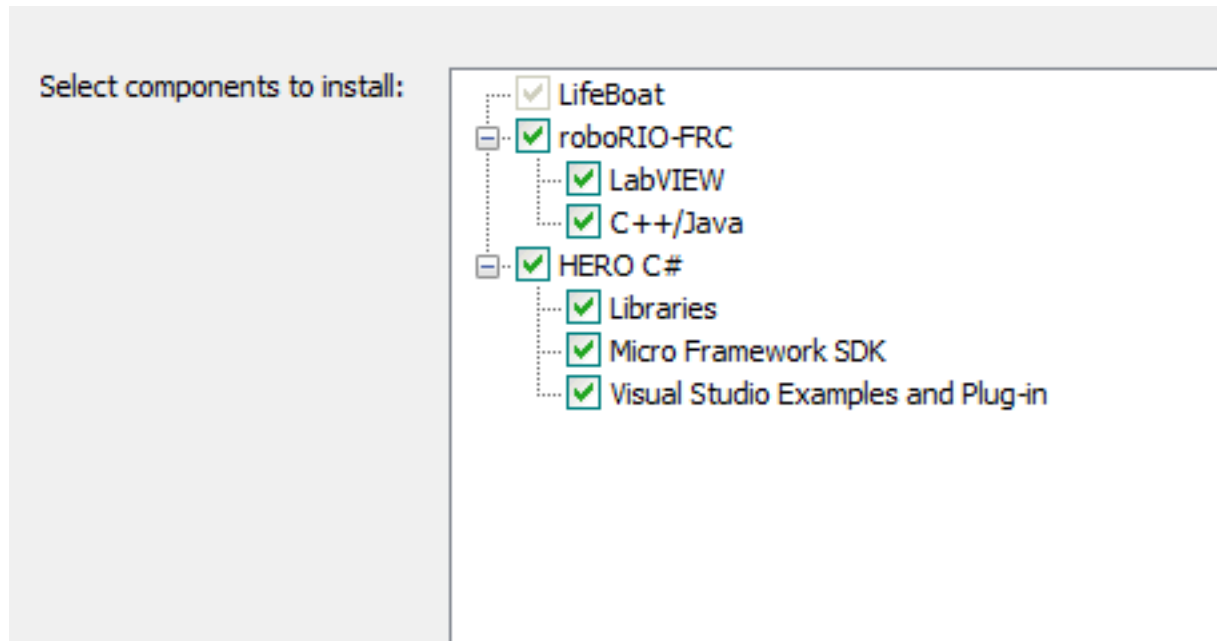
This will look very similar to previous installers - make sure you have the relevant component selected for your programming language.



LV Teams: Make sure LabVIEW is selected. If it is grayed out, then LabVIEW was not installed on the PC.

C++/Java Teams: Make sure C++/Java is selected.

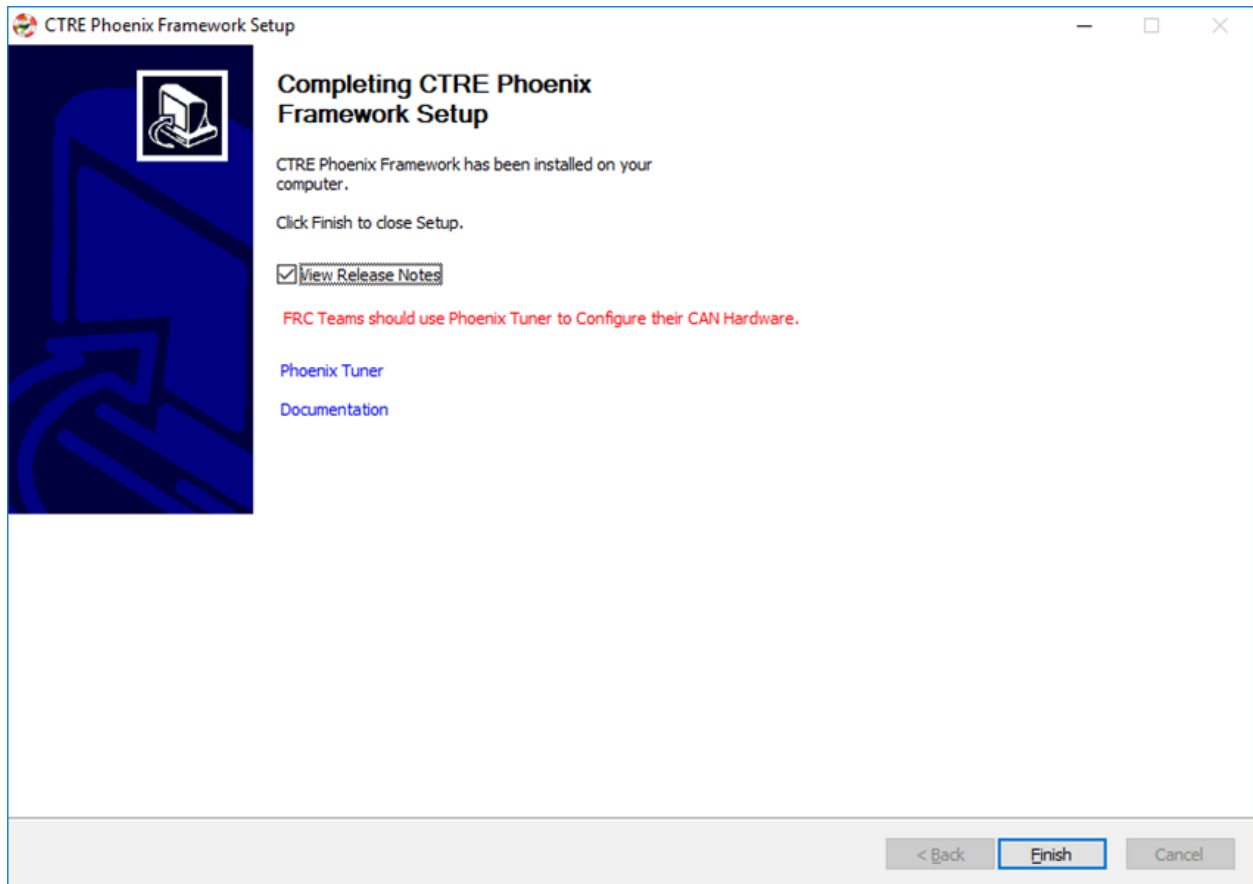
If Visual Studio 2017 (Community/Professional) is detected, HERO C# will be selected. This can be turned off to speed up the installer.



Installer can take anywhere from 30 seconds to 5 minutes depending on which Microsoft runtimes need to be installed.



Final page will look like this. The Phoenix Tuner link can be used to open Phoenix Tuner. Alternatively, you can use the Windows Start Menu.



12.3.2 Option 2: Non-Windows Zip (C++/Java)

The zip will contain **two folders**, “**maven**” and “**vendordeps**”. These folders are meant to be **inserted into your frc2020 install folder**.

See WPI documentation for typical location. <https://docs.wpilib.org/en/latest/docs/software/wpilib-overview/3rd-party-libraries.html#the-mechanism-c-java>

Copy/paste the maven and vendordeps folder into frc2020 folder. This will override a pre-existing Phoenix installation if present.

Note: This will not install Phoenix Tuner or firmware files. If these are necessary (and they typically are) these can be downloaded separately or consider using the complete Phoenix Installer.

12.4 Post Installation Steps

After all workstation installs, the following checks should be followed to confirm proper installation.

12.4.1 FRC C++/Java - Verify Installation

The offline files for vscode are typically installed in:

```
C:\Users\Public\wpilib\2020\vendordeps\Phoenix.json (File used by vscode to include  
→ Phoenix in your project)  
C:\Users\Public\wpilib\2020\maven\com\ctre\phoenix (multiple maven-style library  
→ files)
```

Your drive letter may be different than “C:”. After running the Phoenix Installer, the instructions to add or update Phoenix in your robot project must be followed.

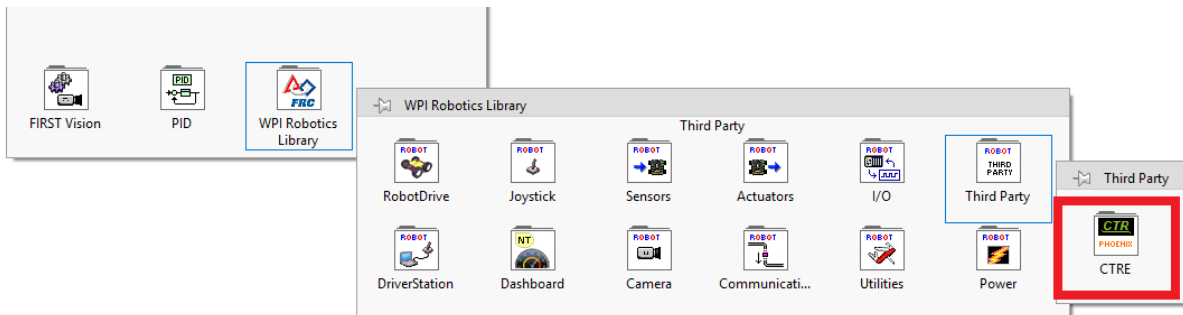
12.4.2 FRC LabVIEW - Verify Installation

After running the installer, open a pristine copy of FRC LabVIEW.

Testing the install can be done by opening LabVIEW and confirming the VIs are installed. This can be done by opening an existing project or creating a new project, or opening a single VI in LabVIEW. Whatever the simplest method to getting to the LabVIEW palette.

The CTRE Palette is located in:

- WPI Robotics Library -> Third Party.

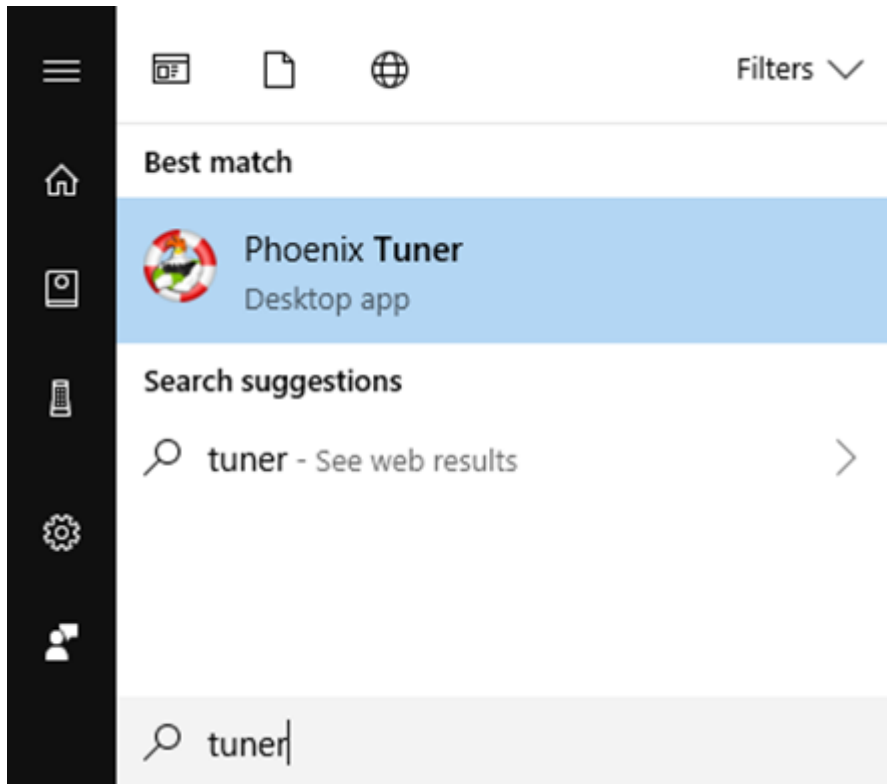


This palette can also be found in:

- WPI Robotics Library -> RobotDrive -> MotorControl -> CanMotor
- WPI Robotics Library -> Sensors -> Third Party
- WPI Robotics Library -> Actuators -> Third Party

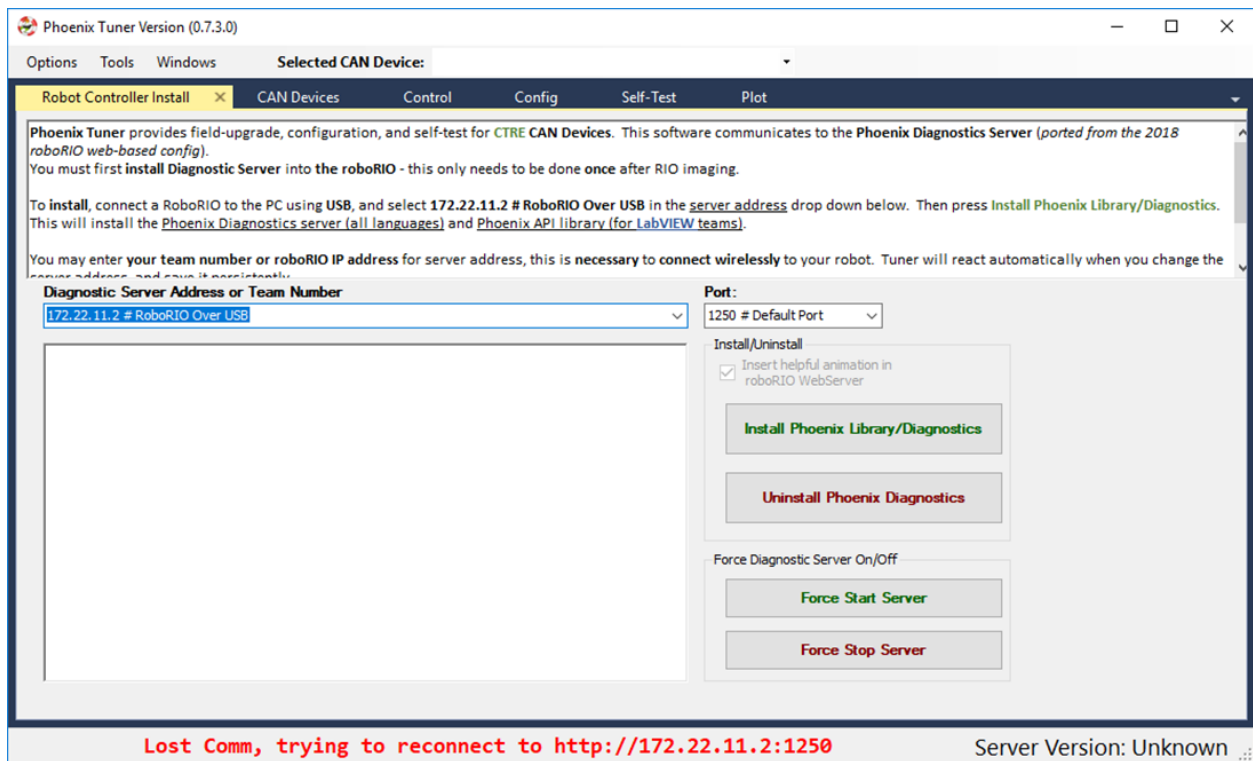
12.4.3 FRC Windows - Open Phoenix Tuner

Open Phoenix Tuner



If this is the first time opening application, confirm the following:

- the status bar should read “Lost Comm”.
- No CAN devices will appear.
- The Server version will be unknown.



FRC: Prepare NI roboRIO

13.1 Why prepare Robot Controller?

In the previous 2019 season, preparing the Robot Controller typically meant:

1. Installing the Phoenix Diagnostics
2. Installing the Phoenix API into roboRIO (if using LabVIEW).

In the 2020 release of Phoenix, both of these are automatically handled by the library deployment features of WPI Visual Studio Code extensions (C++/Java) and NI LabVIEW.

Phoenix Diagnostics has become a library that is compiled into the FRC robot application. This is a result of the roboRIO CAN bus changes implemented by the NI for 2020. Tuner now communicates with “Phoenix Diagnostic Server” running in the deployed application via an HTTP API.

If the roboRIO does not have a deployed application, a temporary Diagnostic Server application can be deployed from Tuner. This is particularly useful during hardware-bringup.

13.1.1 LabVIEW

NI LabVIEW supports a feature that will automatically deploy the Phoenix API libraries to the roboRIO. After running the installer, 2020 LabVIEW robot projects will automatically install Phoenix into the roboRIO when the program is permanently deployed via “Run As Startup”.

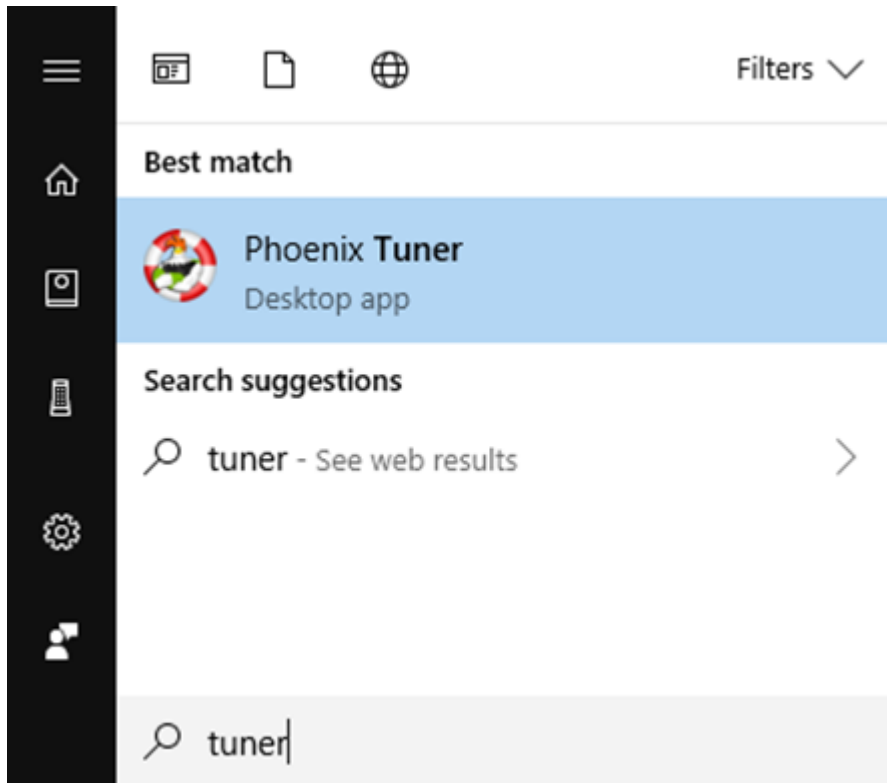
The steps for first deploy are:

1. “Build” the FRC Boot-up Deployment
2. “Run as Startup”
3. Re-boot the roboRIO (see note below)

Note: After first Run-as-Startup (since imaging the RIO), you may see an error in the Driver Station reporting that the Phoenix libraries are missing. A reboot of the RIO will likely resolve this. We recommend using the “Restart roboRIO” button in the Driver Station.

13.2 How to prepare Robot Controller

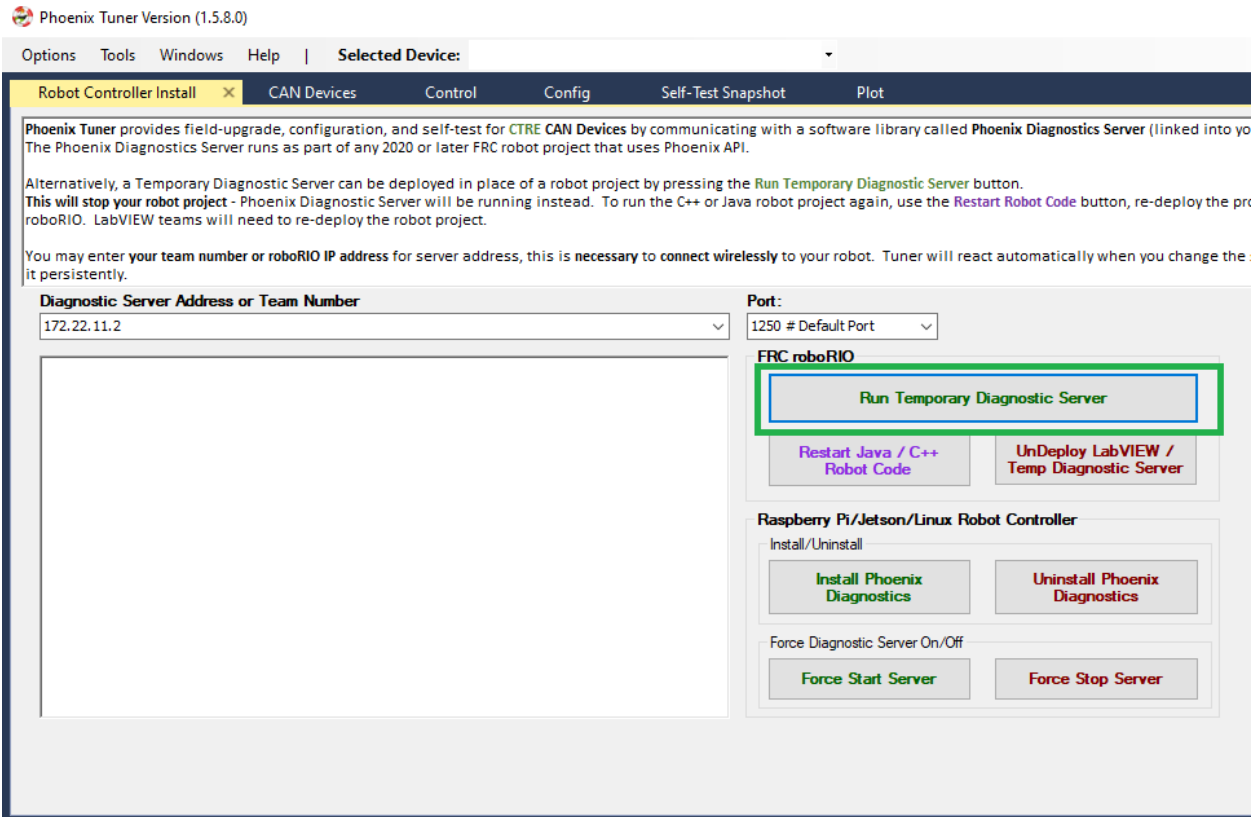
Open Tuner and connect USB between the workstation and the roboRIO.



Select **172.22.11.2 # RoboRIO Over USB** and **1250** for the **address** and **port**. These are generally selected by default, and typically do not require modification.

Deploy the Temporary Diagnostic Server.

Note: This is unnecessary if a robot application has been deployed already (C++, Java, or LabVIEW).



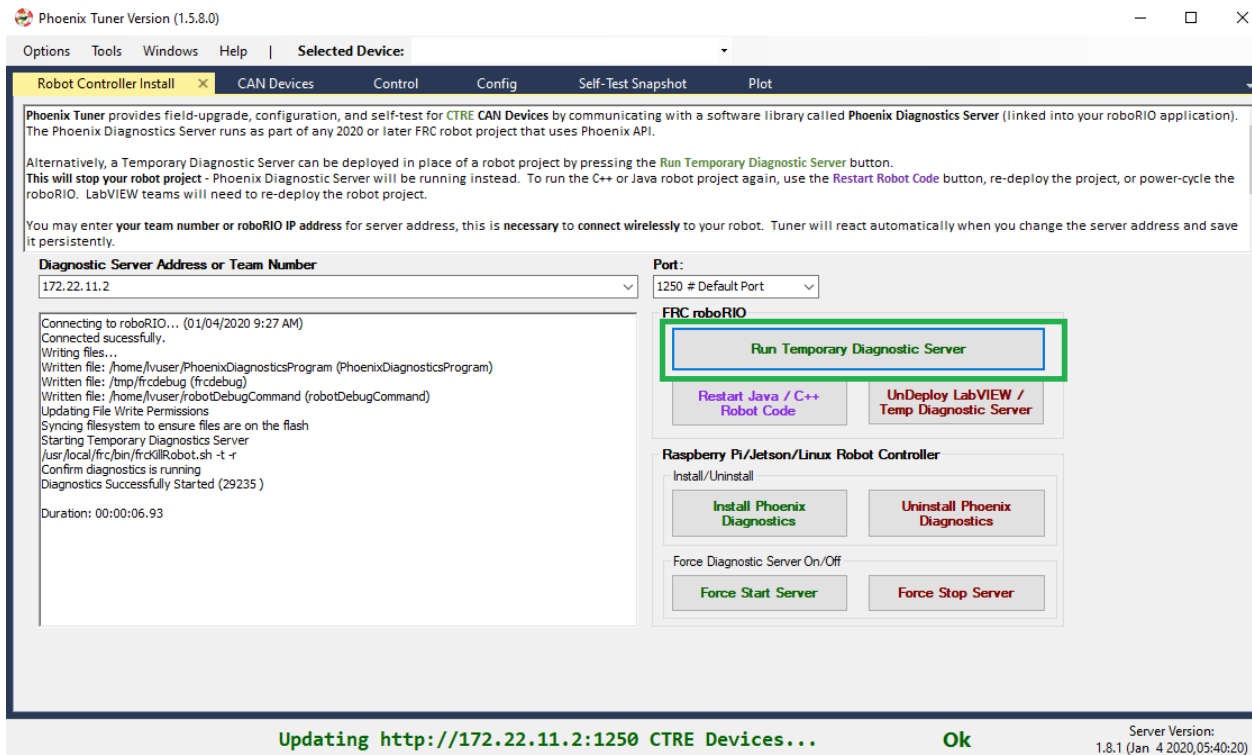
and RIO boot complete? Lost Comm, trying to reconnect to http://10.35.39.2:1250 GeneralError

13.3 Verify the robot controller - Tuner

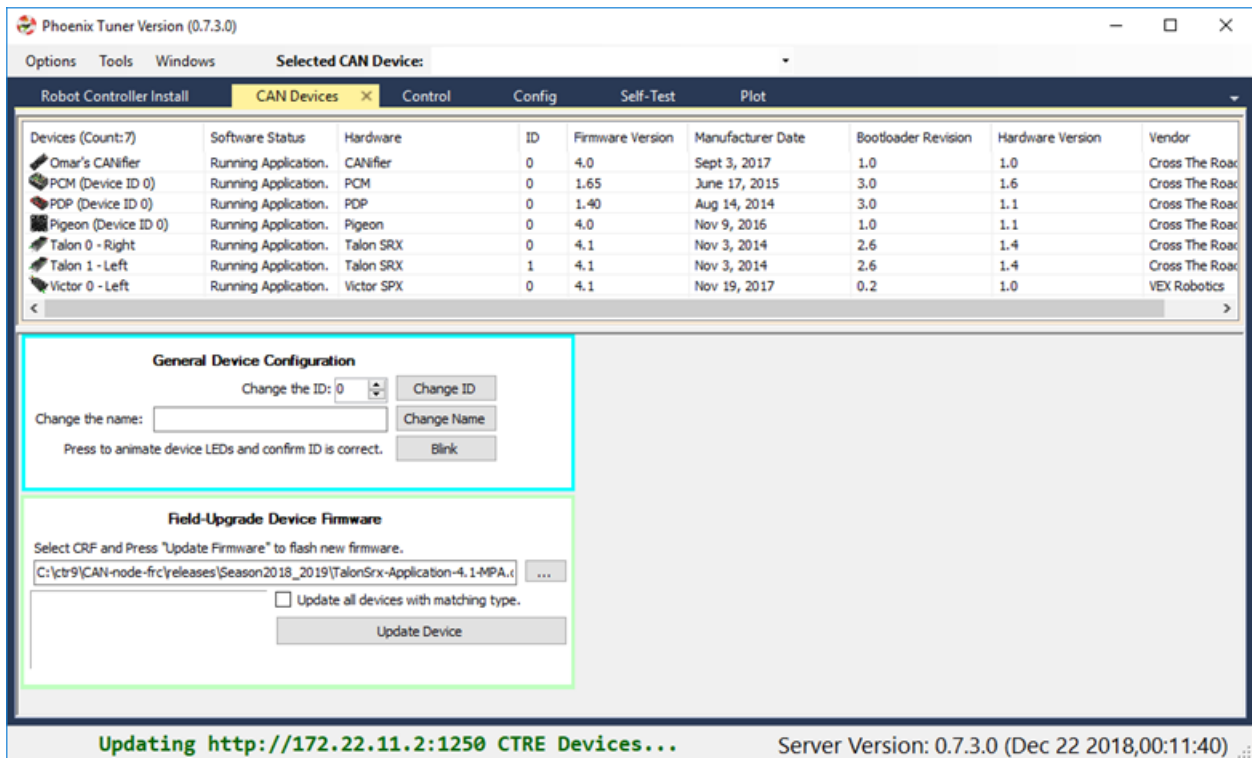
After application deployment, Tuner will immediately connect to the roboRIO.

Confirm the bottom status bar is green and healthy, and server version is present.

FIRST Robotics Competition



If there are CAN device present, they will appear. However, it is possible that devices are missing, this will be resolved in the next major section (CAN Bus bring up).



13.3.1 roboRIO Connection (Wi-Fi/Ethernet)

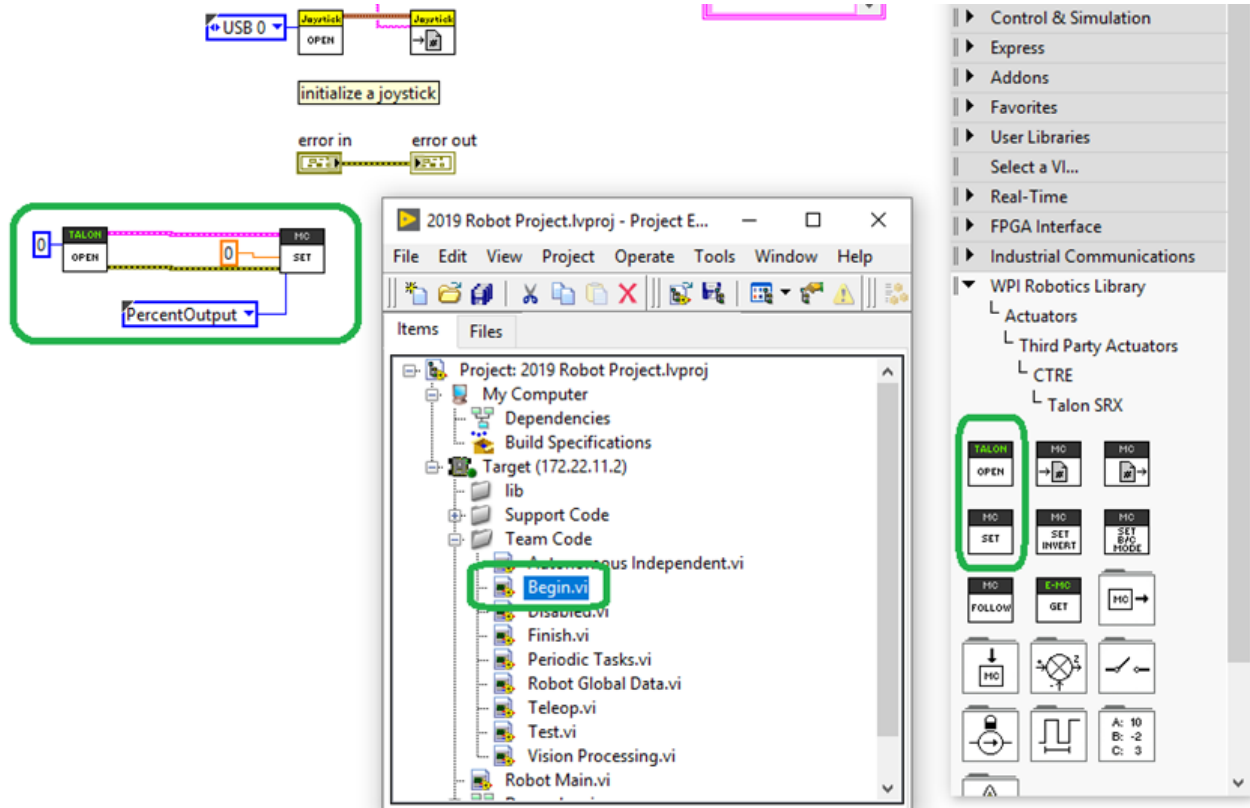
The recommended connection method for control/plotter features is over **USB or using static IP (Ethernet/Wi-Fi)**. The mDNS strategy used by the roboRIO can *sometimes* fail intermittently which can cause hiccups when submitting HTTP requests to the roboRIO.

Testing has shown that using USB (172.22.11.2) or using static IP address has yielded a greater user experience than the roboRIO-team-frc.local host name has.

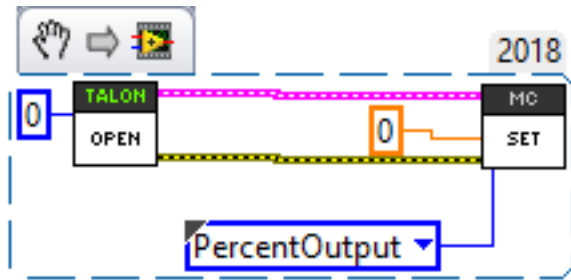
Note: Future releases may have improvements to circumvent the limitations of mDNS.

13.4 Verify the robot controller - LabVIEW

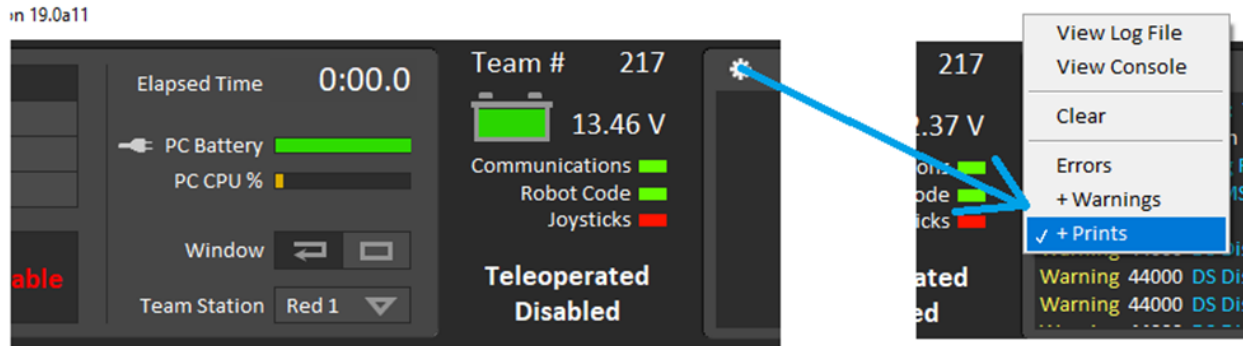
Create a pristine LabVIEW application. Add a CTRE device to Begin.Vi. For example, create a Talon SRX object, even if the device is not physically present.



Tip: Drag drop the following into your Begin.vi



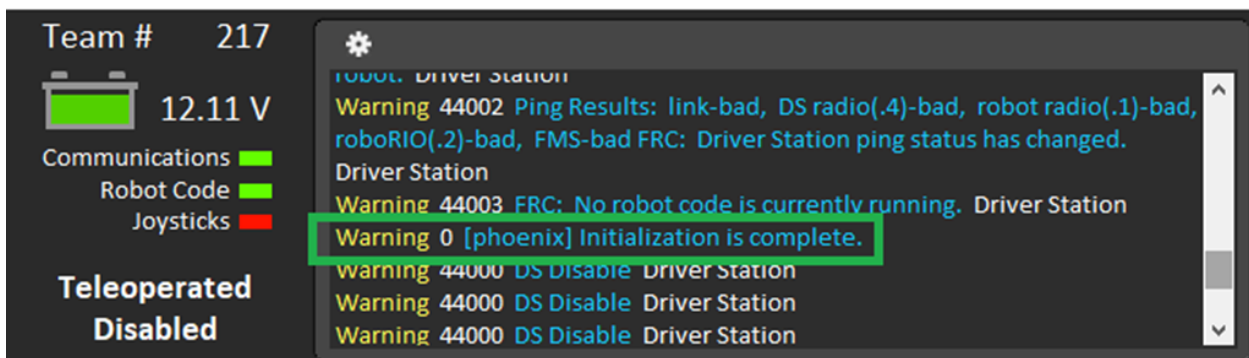
Connect DS and turn on Warnings and Prints by selecting the bottom most option.



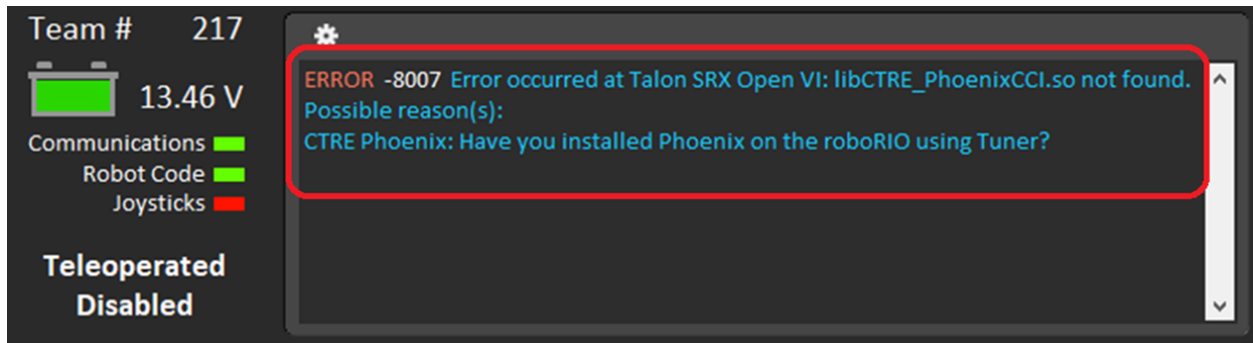
Upload the application to the robot controller and check the driver station message log.

If everything is working, the Phoenix Initialization message can be found.

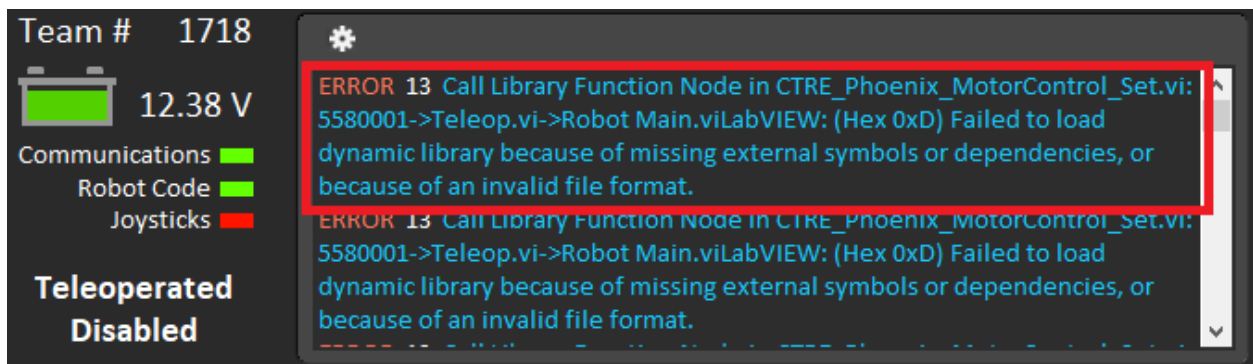
Note: This message will not appear after subsequent “soft” deploy (LabVIEW RAM-only temporary deploys).



If Phoenix API has not been installed into the robot controller, this message will appear.



If you have used Phoenix LifeBoat (which should NOT be used), this message will appear. If this occurs you will need to re-image your roboRIO and then re-follow the instructions in this section exactly, without using LifeBoat.



13.5 Verify the robot controller - Web page

The Silverlight web interface provided in previous seasons is **no longer available**. Moving forward, the NI web interface will likely be much simpler.

As a result, **Phoenix Tuner** may embed a *small message reminder indicating that CAN features have been moved to Tuner*. This will depend on the version of Phoenix.

Typically, the message will disappear after 5 seconds. This will not interfere with normal web page features (IP Config, etc.).

172.22.11.2: System Configuration x +

← → ↻ ⓘ Not secure | 172.22.11.2/#!/SystemConfig

172.22.11.2: System Configuration

Save CAN Bus features have been moved to **Phoenix Tuner 4**

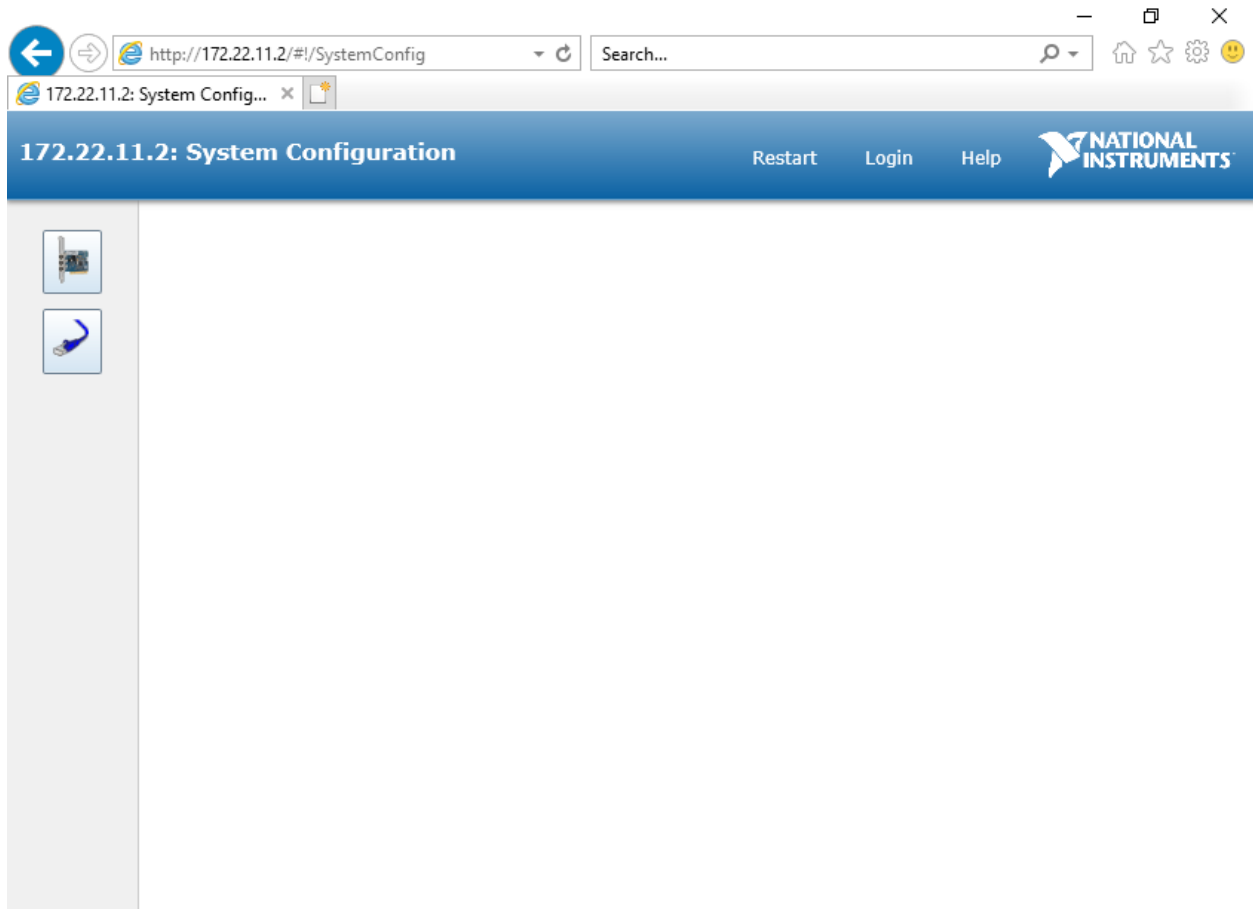
Settings

Hostname	NI-roboRIO-030498A1
IP Address	10.2.17.2 (Ethernet) 172.22.11.2 (Ethernet)
DNS Name	NI-roboRIO-030498A1-2.local
Vendor	National Instruments
Model	roboRIO
Serial Number	030498A1
Firmware Version	6.0.0f1
Operating System	NI Linux Real-Time ARMv7-A 4.9.47-rt37-ni-6.0.0f1
Status	Running
System Start Time	Sat Dec 22 2018 18:56:48 GMT-0500 (Eastern Standard Time)
Image Title	roboRIO Image
Image Version	FRC_roboRIO_2019_v9
Comments	asdf
Locale	English
VISA Resource Name	system

Update Firmware

Warning: The roboRIO Web-page does not provide CAN bus support any more as this has been removed by NI. Use Phoenix Tuner instead.

Warning: The roboRIO Web-page does not render correctly if using Internet Explorer (see below). Recommended browsers are Chrome or Firefox.



13.6 Verify the robot controller - HTTP API

Tuner leverages the HTTP API provided by Phoenix Diagnostics Server.

So technically you have already confirmed this is working.

But, it is worth noting that this HTTP API can potentially be used by third-party software, or even the robot application itself.


Here is a simple get version command and response.

```
http://172.22.11.2:1250/?action=getversion
```



Here is a simple getdevices command and response.

```
http://172.22.11.2:1250/?action=getdevices
```



```

{
  "DeviceArray": [
    {
      "BootloaderRev": "0.2",
      "CurrentVers": "4.1",
      "DynID": 17102859,
      "HardwareRev": "1.0",
      "ID": 17103872,
      "ManDate": "Nov 19, 2017",
      "Model": "Victor SPX",
      "Name": "Victor 0 - Left",
      "SoftStatus": "Running Application.",
      "UniqID": 5,
      "Vendor": "VEX Robotics"
    },
    {
      "BootloaderRev": "2.6",
      "CurrentVers": "4.1",
      "DynID": 33880073,
      "HardwareRev": "1.4",
      "ID": 33881088,
      "ManDate": "Nov 3, 2014",
      "Model": "Talon SRX",
      "Name": "Talon 0 - Right",
      "SoftStatus": "Running Application.",
      "UniqID": 4,
      "Vendor": "Cross The Road Electronics"
    },
    {
      "BootloaderRev": "2.6",
      "CurrentVers": "4.1",
      "DynID": 33880071,
      "HardwareRev": "1.4",
      "ID": 33881089,
      "ManDate": "Nov 3, 2014",
      "Model": "Talon SRX",
      "Name": "Talon 1 - Left",
      "SoftStatus": "Running Application.",
      "UniqID": 6,
      "Vendor": "Cross The Road Electronics"
    },
    {
      "BootloaderRev": "1.0",
      "CurrentVers": "4.0",
      "DynID": 50657290,
      "HardwareRev": "1.0",
      "ID": 50658304,
      "ManDate": "Sept 3, 2017",

```


Initial Hardware Testing

For your competition team to have the best chance of success, hardware components should be tested as soon as they are received. This is generally done by:

- Powering up the device and confirming LED states.
- Ensuring hardware shows up in Tuner if wired to CAN Bus.
- Drive outputs / drive motor in both directions (if motor controller).

This is explained in the sections below, but it is worth pointing out how important this step is. It is in your team's best interest to test ALL purchased robot components immediately and in isolation. Here are the reasons why:

- Robot *replacement* components should be in a **state of readiness**. Otherwise a replacement during competition can yield erroneous behavior.
- Many robot components (in general) have **fixed warranty periods**, and replacements must be done within them.
- Confirming devices are functional **before handing them to students** ensures best chance of success. If a student later damages hardware, they need to understand how they did it to ensure it does not happen again. Without initial validation, you can't determine root-cause.

Much of this is done during the "bring-up" phase of the robot. However, there is much validation a team can do long before the robot takes form.

Unfortunately, there are **many** teams that do not perform this step, and end up isolating devices and re-implementing their cable solutions at competition, because this was not done during robot bring up.

Note: "Bring up / Board bring up / Hardware bring up" is an engineering colloquial phrase. It is the initial setup and validation phase of your bench or robot setup.

Bring Up: CAN Bus

Now that all of the software is installed and verified, the next major step is to setup hardware and firmware.

15.1 Understand the goal

At this point we want to have reliable communication with CAN devices. There are typically two failure modes that must be resolved:

- There are same-model devices on the bus with the same device ID (devices have a default device ID of '0').
- CAN bus is not wired correctly / robustly.

This is why during hardware validation, you will likely have to isolate each device to assign a unique device ID.

Note: CTRE software has the ability to resolve device ID conflicts without device isolation, and CAN bus is capable of reporting the health of the CAN bus (see Driver Station lightening tab). However, the problem is when **both** root-causes are occurring at the same time, this can confuse students who have no experience with CAN bus systems.

Note: Many teams will preassign and update devices (Talon SRXs for example) long before the robot takes form. This is also a great task for new students who need to start learning the control system (with the appropriate mentor oversight to ensure hardware does not get damaged).

Note: Label the devices appropriately so there is no guessing which device ID is what. Don't have a label maker? Use tape and/or Sharpie (sharpie marks can be removed with alcohol).

Warning: Talon SRX and Talon FX must use unique device IDs for Phoenix API to function correctly. This design decision was made so that teams could use the existing TalonSRX class for control.

15.2 Check your wiring

Specific wiring instructions can be found in the user manual of each product, but there are common steps that must be followed for all devices:

- If connectors are used for CANBus, **tug-test each individual crimped wire** one at a time. Bad crimps/connection points are the most common cause of intermittent connection issues.
- Confirm red and black are not flipped. **Motor Controllers typically are not reverse power protected.**
- Confirm battery voltage is adequate (through Driver Station or through voltmeter).
- Manually inspect and confirm that green-connects-to-green and yellow-connects-to-yellow at every connection point. **Flipping/mixing green and yellow is a common failure point during hardware bring up.**
- Confirm breakers are installed in the PDP where appropriate.
- Measure resistance between CANH and CANL when system is not powered (should measure $\sim 60\Omega$). If the measurement is 120Ω , then confirm both RIO and PDP are in circuit, and PDP jumper is in the correct location.

15.3 Power up and check LEDs

If you haven't already, power up the platform (robot, bench setup, etc.) and confirm LEDs are illuminated (at all) on all devices.

You may find many of them are blinking or "blipping" red (no communication).

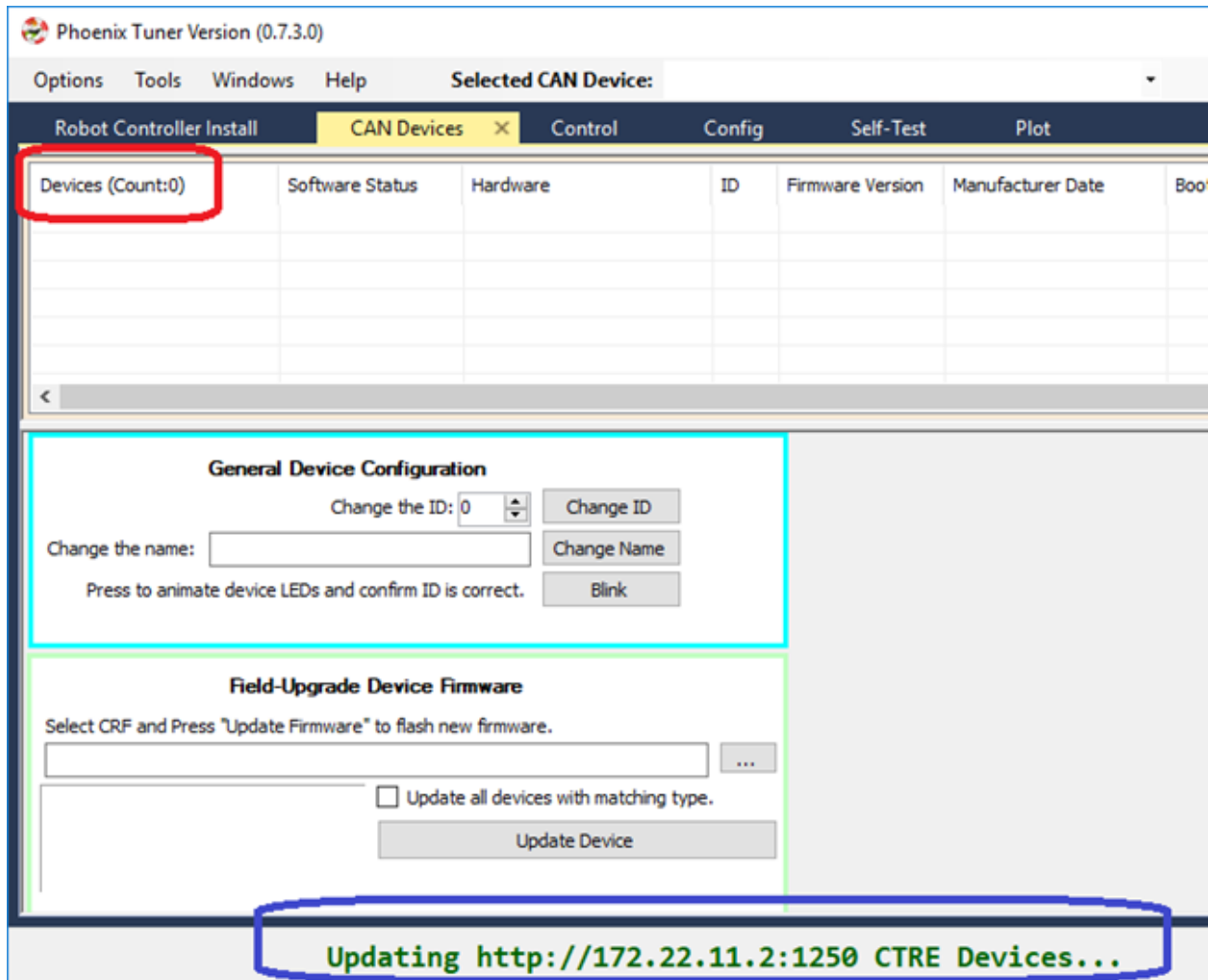
Tip: If you are color-blind or unable to determine color-state, grab another team member to assist you.

Note: If using Ribbon cabled Pigeon IMUs, Pigeon LEDs will reflect the ribbon cable, not the CAN bus. At which point any comm issue with Pigeon will be resolved under section Bring Up: Pigeon IMU.

15.4 Open Phoenix Tuner

Navigate to the CAN devices page.

This capture is taken with no devices connected to the roboRIO. roboRIO will take around 30 seconds to boot.



15.5 LEDs are red - now what?

We need to rule out same-id versus bad-bus-wiring. There are two approaches. Approach 1 will help troubleshoot bad wiring and common IDs. Approach 2 will only be effective in troubleshooting common IDs. But this method is noteworthy because it is simple/quick (no wiring changes, just pull breakers).

The specific instructions for changing device ID are in the next section. Review this if needed.

15.5.1 Approach 1 (best)

Procedure:

- **Physically connect CAN bus from roboRIO to one device only. Circumvent your wiring if need be.**
- Power boot robot/bench setup.

- Open Phoenix Tuner and wait for connection (roboRIO may take ~30 seconds to boot)
- Open CAN devices tab
- Confirm if CAN device appears.
- Use Tuner to change the device ID
- Label the new ID on the physical device
- Repeat this procedure for every device, one at a time.

If you find a particular device where communication is not possible, scrutinize device's power and CAN connection to roboRIO. Make the test setup so simple that the only failure mode possible is within the device itself.

Note: Typically, there must be two termination resistors at each end of the bus. One is in the RIO and one is in the PDP. But during bring-up, if you keep your harness short (such as the CAN pigtail leads from a single Talon) then the internal resistor in the RIO is adequate.

15.5.2 Approach 2 (easier)

Procedure:

- **Leave CAN bus wiring as is.**
- **Pull breakers and PCM fuse from PDP.**
- **Disconnect CAN bus pigtail from PDP.**
- **Pick the first device to power up and restore breaker/fuse/pigtail so that only this CAN device is powered.**
- Power boot robot/bench setup.
- Open Phoenix Tuner and wait for connection (roboRIO may take ~30 seconds to boot)
- Open CAN devices tab
- Confirm if CAN device appears. If device does not appear, scrutinize device's power and CAN connection to roboRIO.
- Use Tuner to change the device ID
- Label the new ID on the physical device
- Repeat this procedure for every device.

If you find a particular device or section of devices where communication is not possible, then the CAN bus wiring needs to be re-inspected. Remember to “flick” / “shake” / “jostle” the CAN wiring in various sections to attempt to reproduce red LED blips. This is a sure sign of loose contact points.

If you are able to detect and change device ID on your devices individually, begin piecing your CAN bus together. Start with roboRIO <—> device <—> PDP, to ensure termination exists at both ends. Then introduce the remaining devices until a failure is observed or until all devices are in-circuit.

If introducing a new device creates a failure symptom, scrutinize that device by replacing it, inspecting common wires, and inspecting power.

Note: If 2014 PDP is the only device that does not appear or has red LEDs, see PDP boot up section for specific failure mode.

Note: If ribbon cable Pigeon does not appear, it likely is because Talon has old firmware.

At the end of this section, all devices should appear (notwithstanding the above notes) and device LEDs should not be red. PCM, Talon, Victor, Pigeon, and CANifier typically blink orange when they are healthy and not controlled. PDP may be orange or green depending on its sticky faults.

15.6 Set Device IDs







Note: A CTRE device can have an ID from 0 to 62. If you select an invalid ID, you will generally get an immediate prompt.

Below we see several devices, however the physical robot has 19 actual devices. Because all the Talons have a device ID of '0', they do not show up as unique hardware. This must be resolved before you can attempt utilizing them.

Phoenix Tuner Version (0.7.3.0)

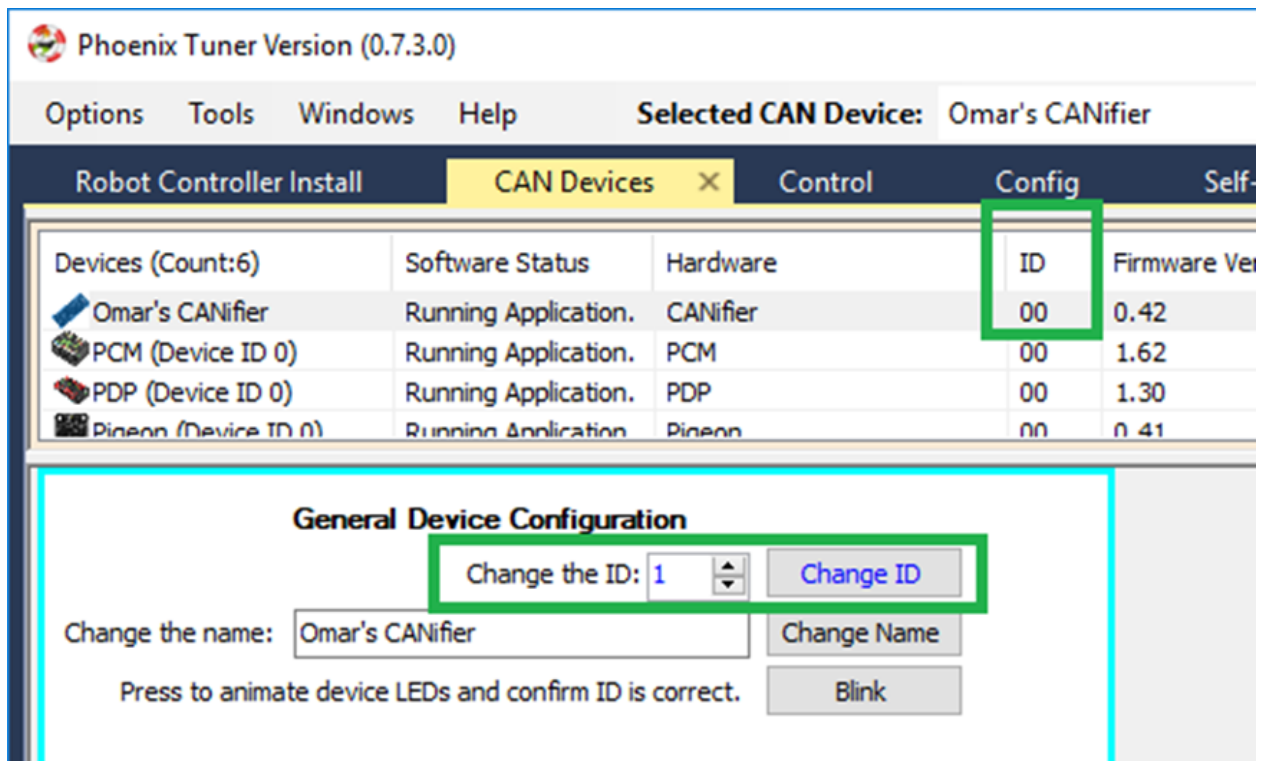
OptionsToolsWindowsHelpSelected CAN Device:

Robot Controller InstallCAN DevicesControlConfigSelf-TestPlot

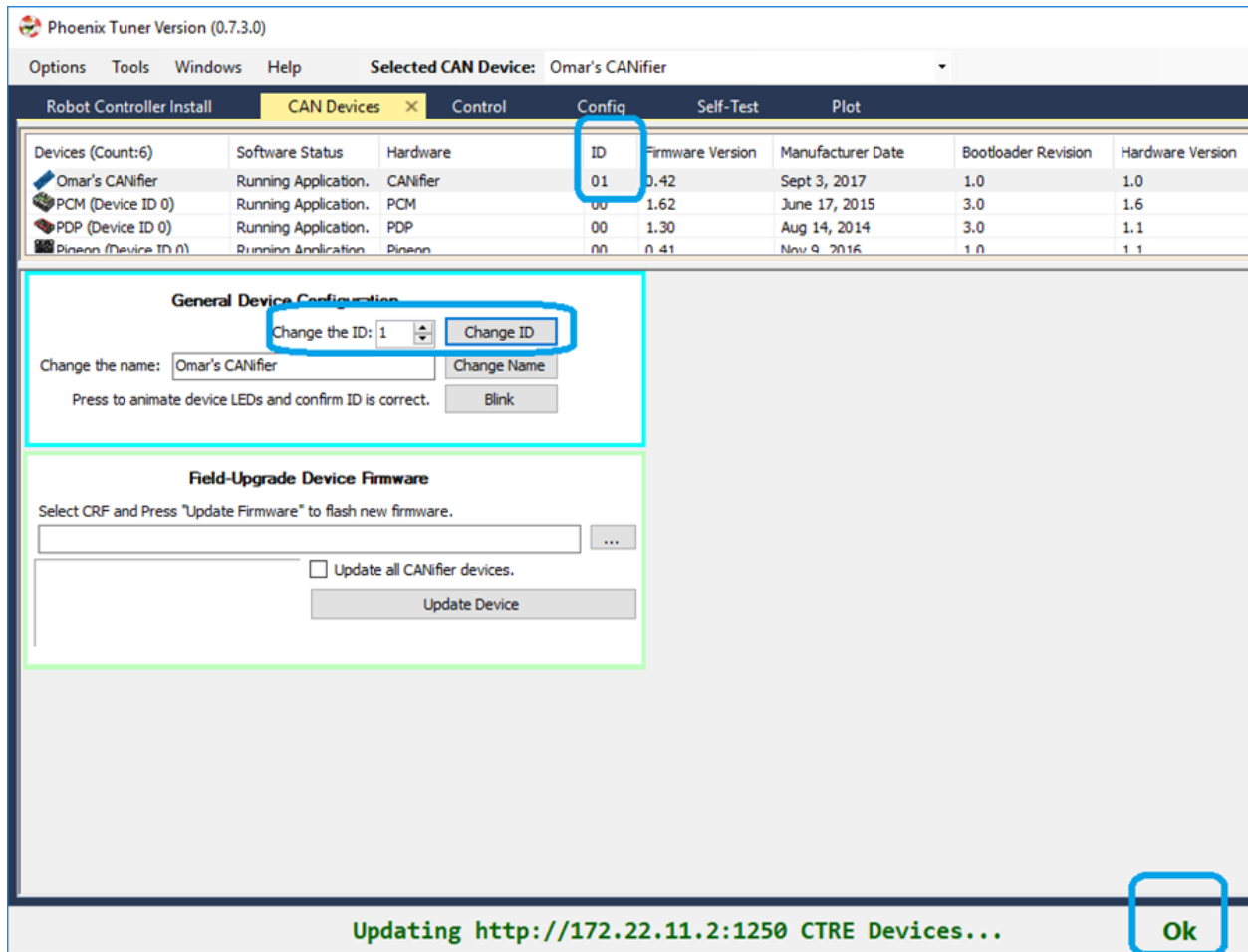
Devices (Count:6)	Software Status	Hardware	ID	Fi...	Manufactur...	B...	H..	Vendor
 Omar's CANifier	Running Application.	CANifier	00	0.42	Sept 3, 2017	1.0	1.0	Cross The Road Electronics
 PCM (Device ID 0)	Running Application.	PCM	00	1.62	June 17, 2015	3.0	1.6	Cross The Road Electronics
 PDP (Device ID 0)	Running Application.	PDP	00	1.30	Aug 14, 2014	3.0	1.1	Cross The Road Electronics
 Pigeon (Device ID 0)	Running Application.	Pigeon	00	0.41	Nov 9, 2016	1.0	1.1	Cross The Road Electronics
 Talon SRX (Device ID 0)	There are 14 devices with this Device ID. Running Application.	Talon SRX	00	3.9	Aug 14, 2015	3.2	1.7	Cross The Road Electronics
 Victor 0 - Left	Running Application.	Victor SPX	00	3.9	Nov 19, 2017	0.2	1.0	VEX Robotics

Note: We recommend isolating each device and assigning a unique ID first. But in the event there is a conflict, expect an entry mentioning multiple devices. When selecting a device, the actually physical device selected will be the conflict-id device that booted last. You can use this information to control which Talon you are resolving by power cycling the conflict device, then changing its ID in Tuner.

Select the device and use the numeric entry to change the ID. Note the text will change blue when you do this. Then press "Change ID" to apply the changes.



If operation completes, an OK will appear in the bottom status bar (this is true of all operations). Also note the ID has updated in the device list, and the ID text is now black again.

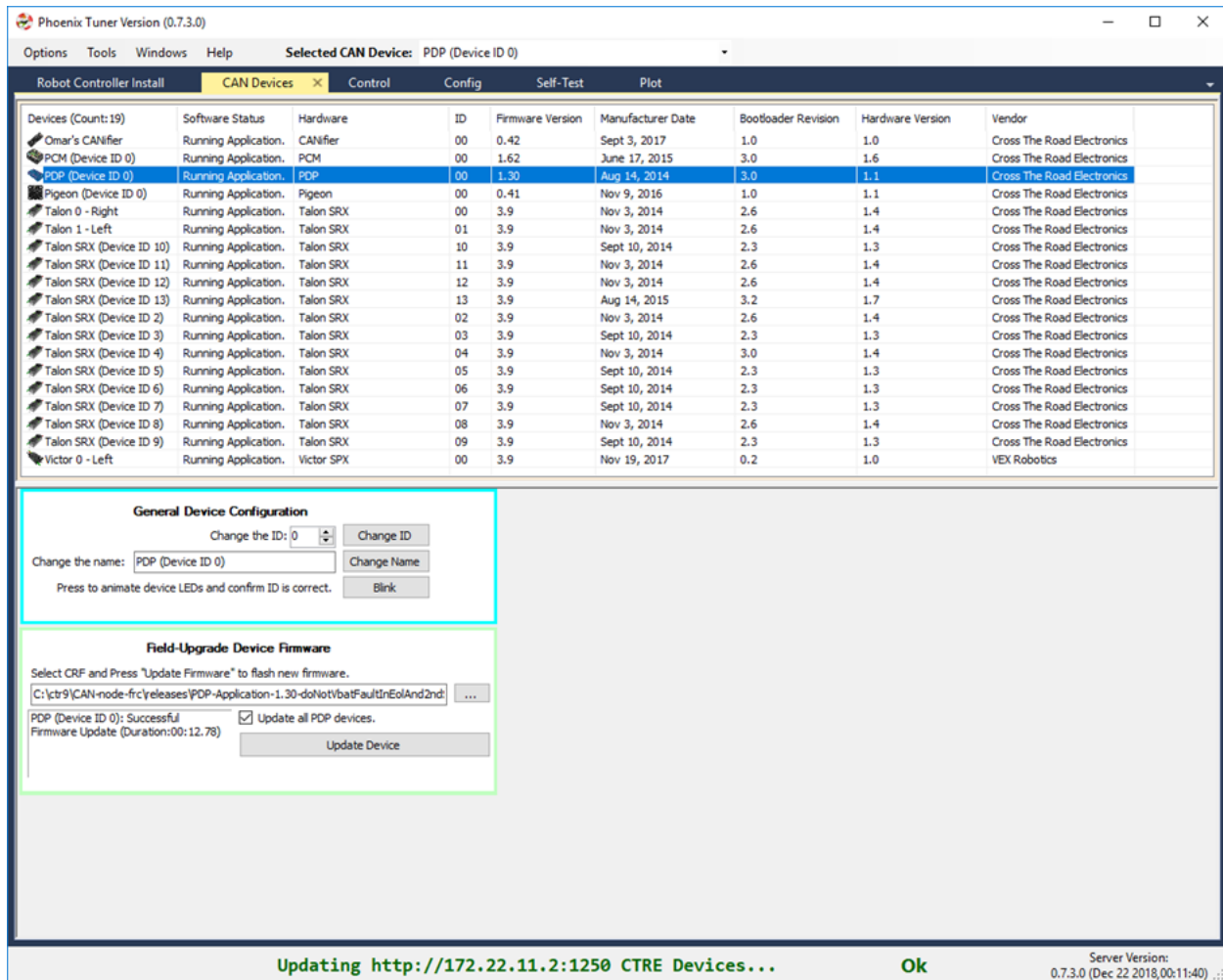


Tip: All production CTRE hardware ships with a default ID of '0'. As a result, it is useful to start your devices at device ID '1', so you can cleanly add another out-of-box device without introducing a conflict.

When complete, make sure every device is visible in the view. Use the Blink button on each device to confirm the ID matches the expected physical device.

Note: The device count is present in the top left corner of the device list. Use this to quickly confirm all devices are present.

Note: If ribbon-cabled pigeon is not present, then the host talon likely has old firmware.



15.7 Field upgrade devices

At this point all devices are present, but the firmware is likely old.

The 2020 season has 20.X firmware for Talon FX, Talon SRX, Victor SPX, CANCoder, CANifier, and Pigeon IMU. Moving forward, the first number of the version will represent the season (next year's 2021 firmware will be 21.X).

20.X firmware is required for all motor controllers and CANCoder. 20.X is also recommended for CANifier and Pigeon IMU.

Note: Latest PDP is 1.40. PDP typically ship with 1.30. 1.40 has all of the signals read by the WPILib software, and will tare the current measures so current will read 0 instead of ~1-2 amps when there is no current-draw. Updating to 1.40 is recommended.

Note: Latest PCM is 1.65. PCM typically ship with 1.62. Firmware 1.65 has an improvement where hardware-revision 1.6 PCMs will not-interrupt compressor when blacklisting a shorted

solenoid channel. Older revisions will pause the compressor in order to safely sticky-fault, new revisions have no need to do this (if firmware is up to date).

The screenshot shows the Phoenix Tuner Version (0.7.3.0) interface. The 'Selected CAN Device' is 'Talon 0 - Right'. The 'CAN Devices' tab is active, displaying a table of devices. Below the table, the 'General Device Configuration' section allows changing the ID and name. The 'Field-Upgrade Device Firmware' section shows a file path for a CRF file and an 'Update Device' button.

Devices (Count: 19)	Software Status	Hardware	ID	Firmware
Omar's CANifier	Running Application.	CANifier	01	0.42
PCM (Device ID 0)	Running Application.	PCM	00	1.62
PDP (Device ID 0)	Running Application.	PDP	00	1.30
Pigeon (Device ID 0)	Running Application.	Pigeon	00	0.41
Talon 0 - Right	Running Application.	Talon SRX	00	3.9
Talon 1 - Left	Running Application.	Talon SRX	01	3.9
Talon SRX (Device ID 10)	Running Application.	Talon SRX	10	3.9

General Device Configuration

Change the ID: 0

Change the name: Talon 0 - Right

Press to animate device LEDs and confirm ID is correct.

Field-Upgrade Device Firmware

Select CRF and Press "Update Firmware" to flash new firmware.

Electronics\LifeBoat\HERO Firmware Files\TalonSrx-Application-4.1-MPA-2019.crf

☒ Update all Talon SRX devices.

Updating <http://172.22.11.2:1250> CTRE Devi

Select the CRF under the Field-upgrade section then press Update Device. The CRFs are available in multiple places, and likely are already on your PC/ See section "Device Firmware Files (crf)".

If there are multiple devices of same type (multiple Talon SRXs for example), you may check Update all devices. This will automatically iterate through all the devices of the same type, and update them. If a device field-upgrade fails, then the operation will complete. Confirm Firmware Version column in the device list after field-upgrade.

Note: Each device takes approximately 15 seconds to field-upgrade.

When complete every device should have latest firmware.

15.8 Pick device names (optional)

The device name can also be changed for certain device types: - CANifier - Pigeon IMU (on CAN bus only) - Talon SRX and Victor SPX

Note: PDP and PCM do not support this.

Note: Ribbon cabled Pigeon IMUs do not support this.

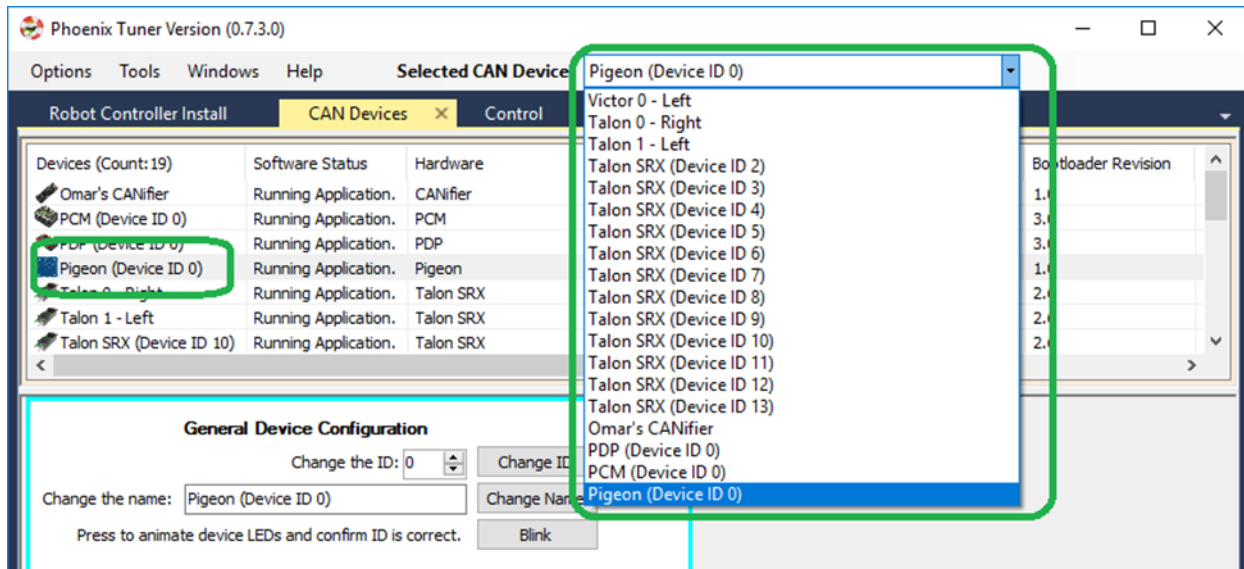
Note: To re-default the custom name, clear the “Name” text entry so it is blank and press “Save”.

15.9 Self-test Snapshot

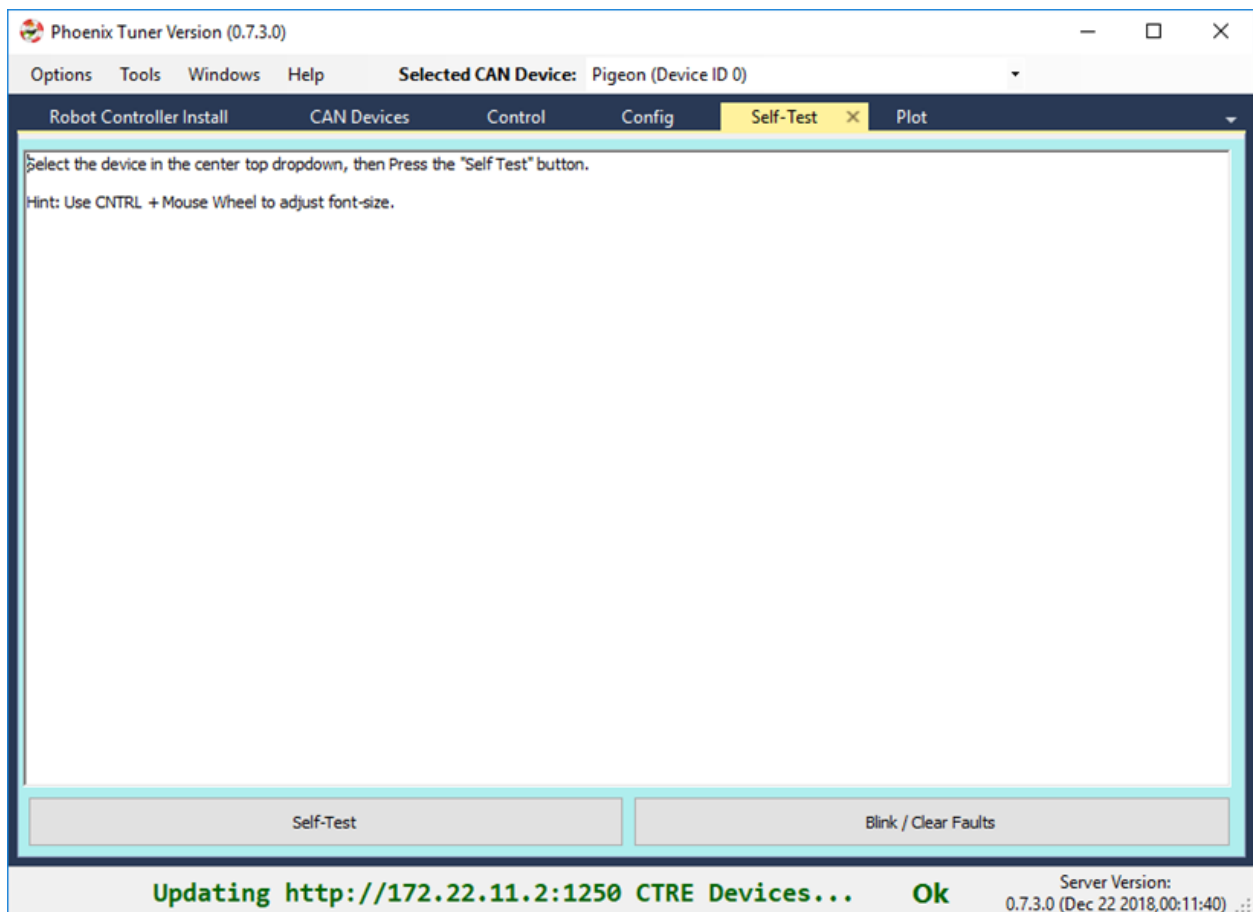
At this point every device should be present on the bus, and updated to latest. This is an opportune time to test the Self-test Snapshot feature of each device.

Select each device either in the device list, or using the dropdown at the center-top. This dropdown is convenient as it is accessible regardless of how the tabs are docked with Tuner.

Note: If you press the “Selected CAN device” text next to dropdown, you will be taken back to the CAN Devices tab.



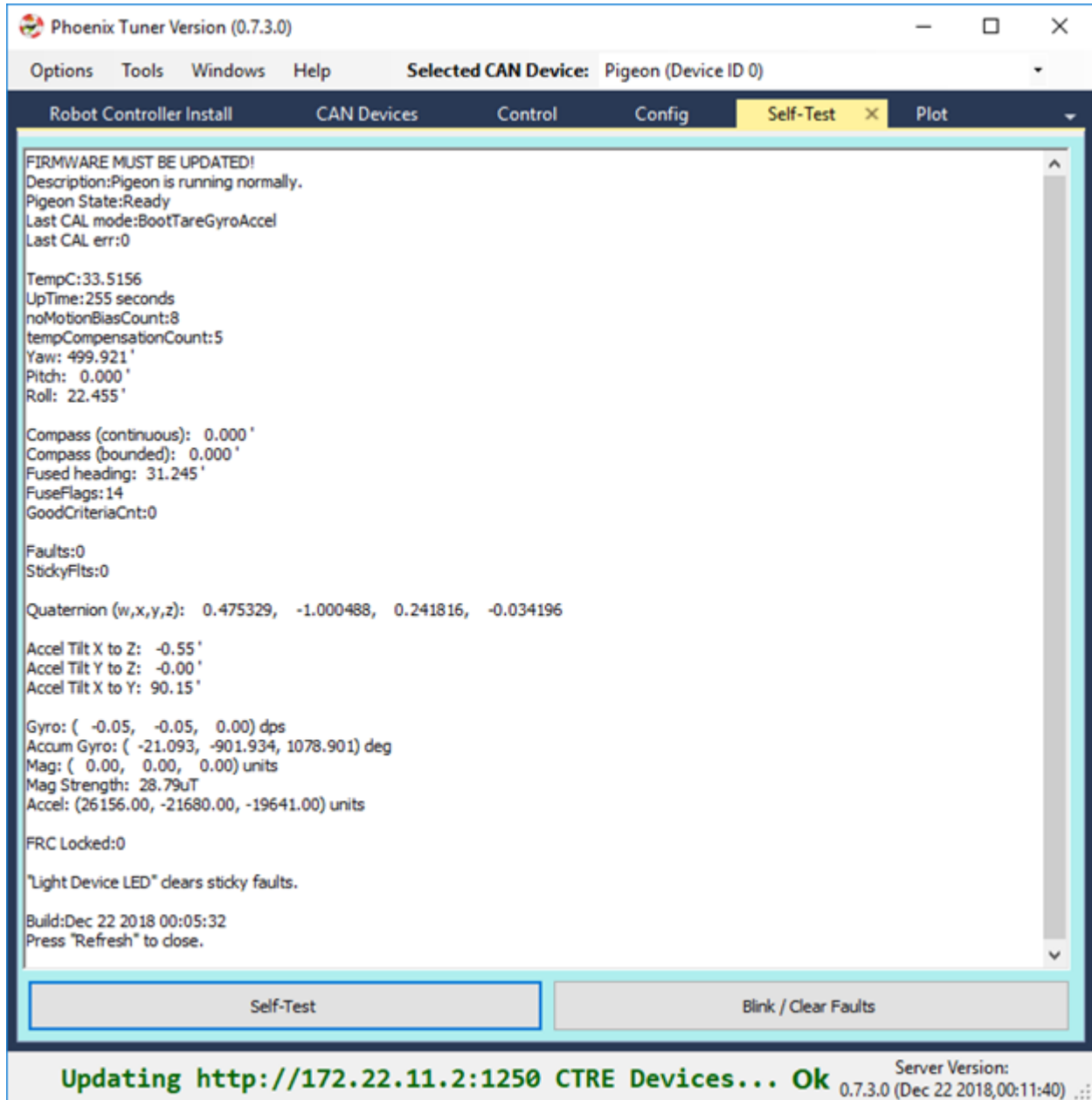
Navigate to the Self-test Snapshot tab. If Self-test Snapshot tab is not present, use the Windows menu bar to reopen it.



Press Self-test Snapshot button and confirm the results.

Note: This Pigeon has not had its firmware updated, this is reported at the top.

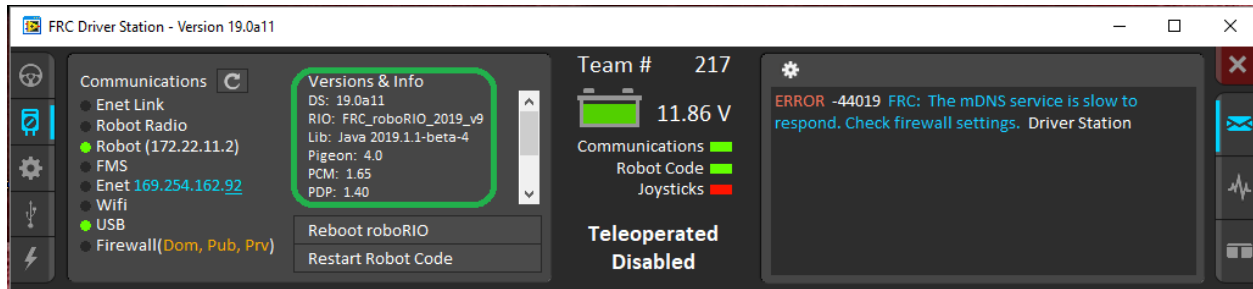
You can also use the Blink/Clear faults button to blink the selected device and clear any previously logged sticky faults.



15.10 Driver Station Versions Page

It is worth mentioning there is basic support of reporting the CAN devices and their versions in the diagnostics tab of the Driver Station.

If there is a mixed collection of firmware versions for a given product type, the version will report “Inconsistent”.



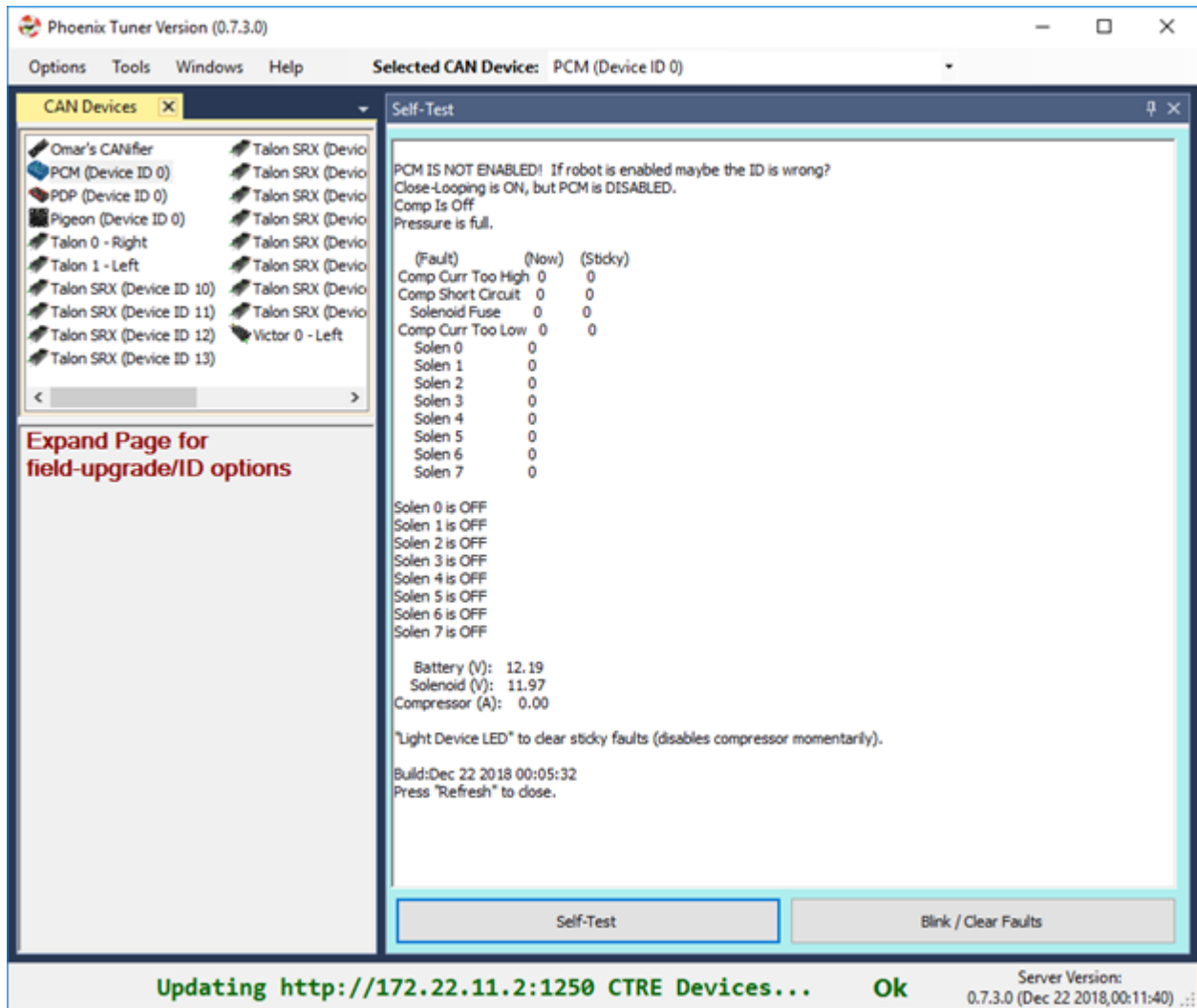
Note: The recommended method for confirming firmware versions is to use Phoenix Tuner.

Note: There is a known issue where ribbon-cabled Pigeons may erroneously report as a Talon. Since this is not a critical feature of the Driver Station, this should not be problematic for FRC teams.

At this point PCM will have firmware 1.62 or 1.65 (latest). Open Phoenix Tuner to confirm.

16.1 Phoenix Tuner Self-test Snapshot

Press Self-test Snapshot to confirm solenoid states, compressor state ,and battery/current measurements. Since device is not enabled, no outputs should assert.



Note: In this view, the Self-test Snapshot was docked to the right. If CAN Devices width is shrunk small enough, the field-upgrade and Device ID options are hidden and the list view becomes collapsed. This way you can still use the device list as an alternative to the center-top dropdown.

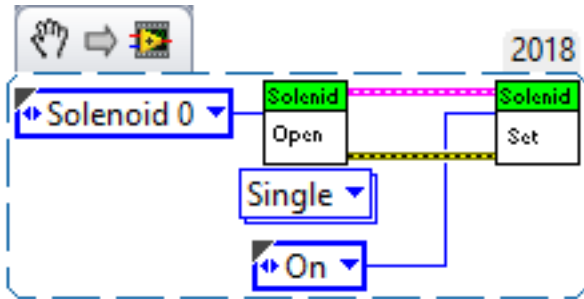
The next step is to get the compressor and solenoids operational.

Create a Solenoid object in LabVIEW/C++/Java and set channel 0 to true.

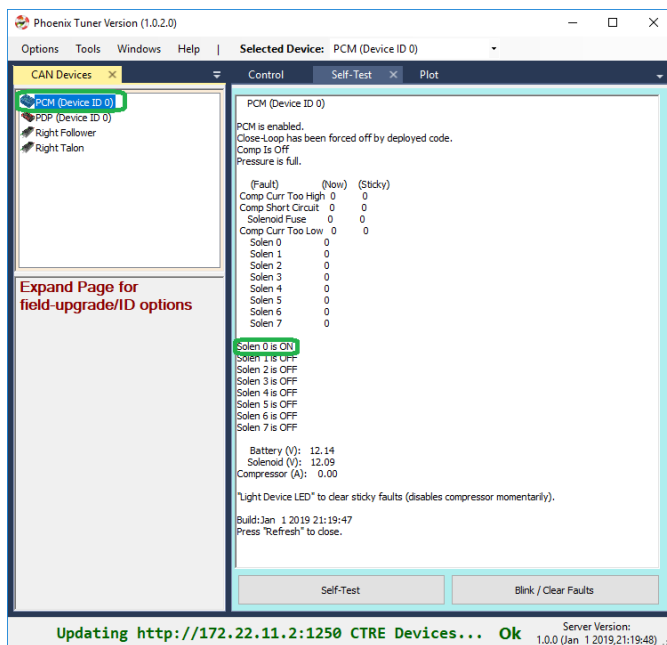
```
import edu.wpi.first.wpilibj.Solenoid;
public class Robot extends TimedRobot {
    Solenoid _solenoid = new Solenoid(0, 0); // first number is the PCM ID (usually 0), second number is the solenoid channel

    public void teleopPeriodic() {
        _solenoid.set(true);
    }
}
```

Tip: Image below can be dragged/dropped into LabVIEW editor.



Then confirm using the Solenoid LED on the PCM and Self-test Snapshot in Tuner.



Generally creating a solenoid object is sufficient for the compressor features to function. In order for the compressor output to activate, all of the following conditions must be met:

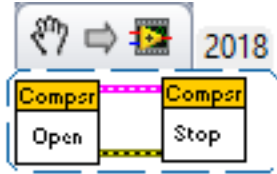
- The robot is enabled via the Driver Station
- Robot application has created a solenoid (or compressor object) with the correct PCM device ID.
- PCM must be powered/wired to CAN Bus.
- Pressure-switch reads too-low (can be confirmed in Self-test Snapshot).
- No compressor related faults occur (can be confirmed in Self-test Snapshot)

Tip: Creating a compressor object is not necessary, but can be useful to force the compressor **off despite pressure reading too-low** with the `setClosedLoopControl` routine/VI. This can be useful for robot power management during critical operations.

```
import edu.wpi.first.wpilibj.Compressor;
public class Robot extends TimedRobot {
    Compressor _compressor = new Compressor();

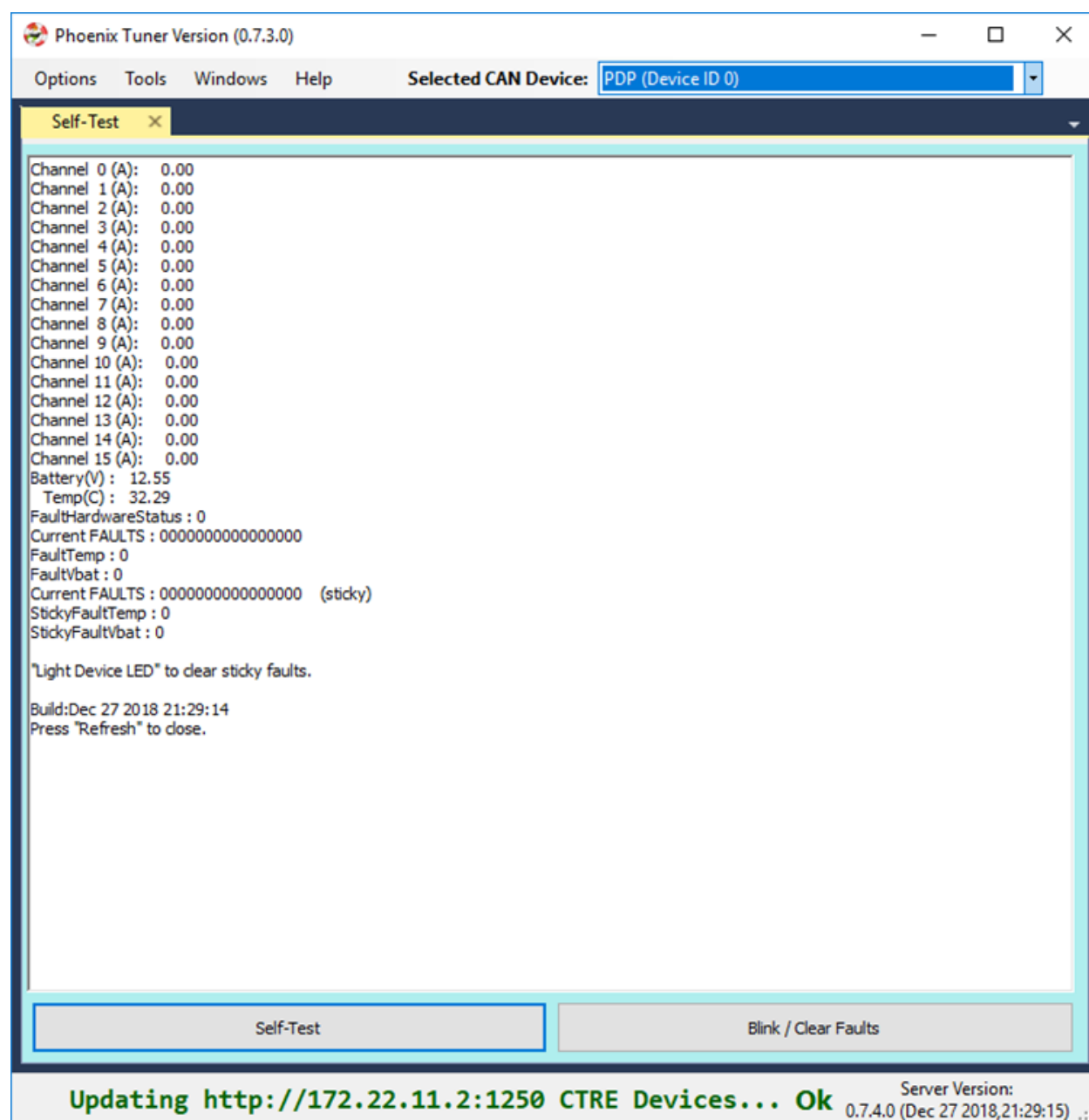
    public void teleopPeriodic() {
        _compressor.setClosedLoopControl(false); //This will force the compressor off
    }
}
```

Tip: Image below can be dragged/dropped into LabVIEW editor.



Bring Up: PDP

At this point PDP will have firmware 1.40 (latest). Open Phoenix Tuner to confirm.
Use Self-test Snapshot to confirm reasonable values for current and voltage.



17.1 Getting sensor data

Sensor data can also be retrieved using the base FRC API available in LabVIEW/C++/Java. See WPI/NI/FRC documentation for how.






17.2 DriverStation Logs

Driver Station logs are automatically generated during normal FRC use. This includes current logging on all PDP Wago channels. Review WPI/NI/FRC documentation for how leverage this.

17.3 2015 Kick off Kit PDPs

There is a known issue with 2015 Kickoff-Kit PDPs where the PDP will not appear on CAN bus and/or LEDs will be red, despite all other devices on the CAN bus functioning properly. This is due to an ESD vulnerability that only exists in the initial manufacture run in 2014. Every season PDP afterwards does not have this issue.

Manufacture date of the PDP can be checked in Tuner. Any PDP with a manufacture date of August 14, 2014 may have this issue. No other PDPs (even those with other 2014 manufacture dates) are known to be affected.

Robot Controller Install		CAN Devices	Control	Config	Self-Test	Plot
Devices (Count: 5)		Software Status	Hardware	ID	Firmware Version	Manufacturer Date
	CANifier (Device ID 0)	Running Application.	CANifier	00	0.40	
	PDP (Device ID 0)	Running Application.	PDP	00	1.40	Aug 14, 2014
	PDP (Device ID 13)	Running Application.	PDP	13	1.30	
	Talon SRX (Device ID 1)	Running Application.	Talon SRX	01	4.11	Nov 3, 2014
	Talon SDV (Device ID 2)	Running Application.	Talon SDV	02	4.11	Aug 14, 2015

These PDPs do correctly provide power and terminate the CAN bus with no compromises. However, the current measurement features may not be correct or available on this version of PDP. If such a PDP is re-used or re-purposed, we recommend using it on your practice robot or for bench setups, and not for competition.

Bring Up: Talon FX/SRX and Victor SPX

At this point all Talon and Victors should appear in Tuner with up to date firmware. The next goal is to drive the motor controller manually. This is done to confirm/test:

- Motor and motor wiring
- Transmission/Linkage
- Mechanism design
- Motor Controller drive (both directions)
- Motor Controller sensor during motion

Note: Talon FX/SRX and Victor SPX can be used with PWM or CAN bus. This document covers the CAN bus use-case.

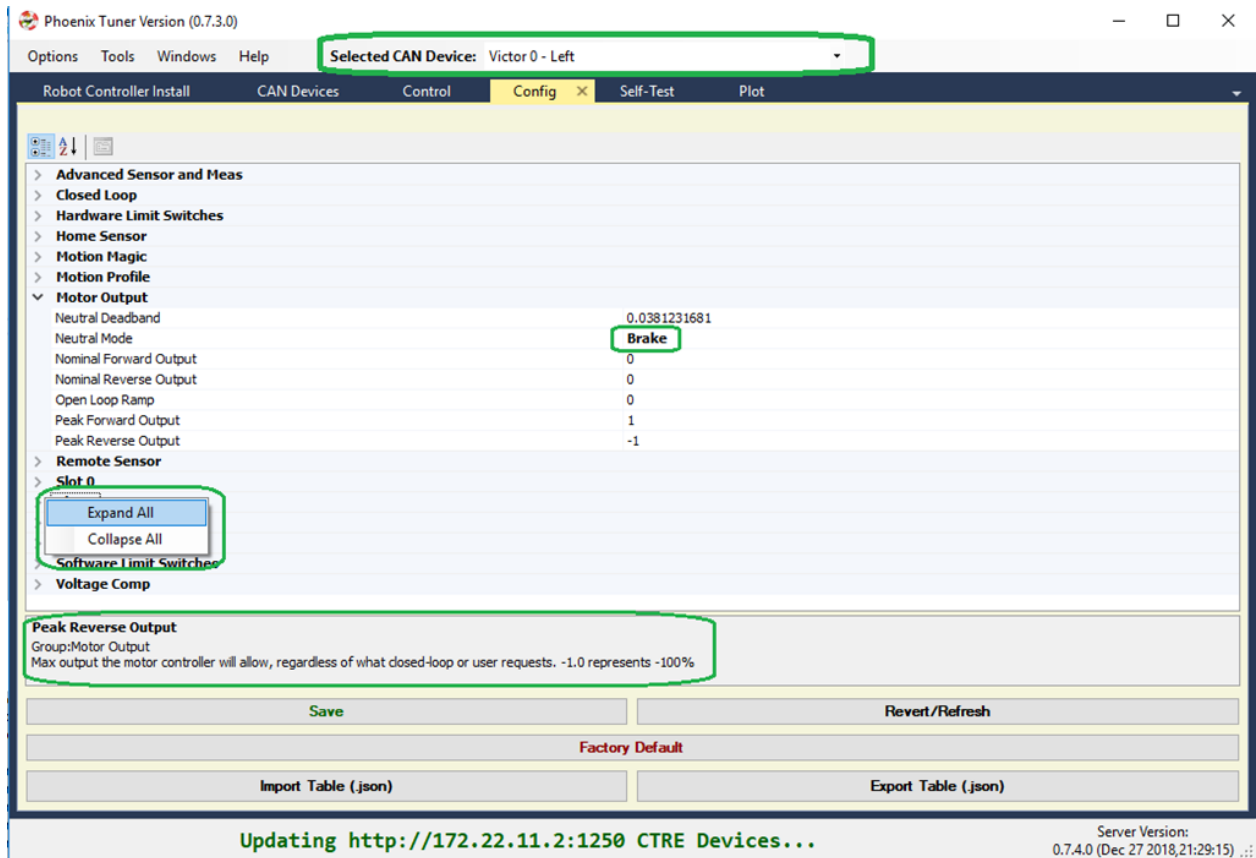
Before we enable the motor controller, first check or reset the configs in the next section.

18.1 Factory Default Motor Controller

Open the config view to see all persistent settings in the motor controller. This can be done in the config tab (Windows => Config).

Select the Victor or Talon in the center-top dropdown. This will reveal all persistent config settings.

Press Factory Default to default the motor controller settings so that it has predictable behavior.



Tip: Right-click anywhere in the property inspector and select Collapse-all to collapse each config group.

Tip: Other configs can be set in this view for testing purposes. For example, you may want to restrict the maximum output for testing via the Peak output settings under “Motor Output”.

Tip: When a setting is modified, it is set to bold to indicate that it is pending. The bold state will clear after you Save.

Tip: If changing a config live in the robot controller, use the Refresh/Revert button to confirm setting in Tuner.

Note: CTRE devices can be factory defaulted via the API, and thru the B/C mechanical button.

Note: Neutral Mode will not change during factory default as it is stored separately from the other persistent configs.

18.2 Configuration

Configurable settings are persistent settings that can be modified via the Phoenix API (from robot code) or via Tuner (Config tab). They can also be factory defaulted using either method.

Configs are modified via the config* routines and LabVIEW Vis. There are two general methods for robust operation of a robot. Additionally you can modify the configs via Tuner.

18.2.1 Method 1 - Use the configAll API

Starting with 2019, there is a single routine/VI for setting all of the configs in a motor controller. This ensures that your application does not need to be aware of every single config in order to reliably configure a fresh or unknown motor controller.

This is the recommend API for new robot projects.

Tip: Config structure/object defaults all values to their factory defaults. This means generally you only need to change the settings you care about.

Tip: When using C++/Java, leverage the IntelliSense (Auto-complete) features of the IDE to quickly discover the config settings you need.

18.2.2 Method 2 - Factory Default and config* routines

Phoenix provides individual config* routines for each config setting. Although this is adequate when the number of configs was small, this can be difficult to manage due to the many features/configs in the CTRE motor controllers.

If using individual config routines, we recommend first calling the configFactoryDefault routine/VI to ensure motor controller is restored to a known state, thus allowing you to only config the settings that you intend to change.

This is recommend for legacy applications to avoid porting effort.

18.2.3 Method 3 - Use Tuner

Tuner can be used to get/set/export/import the configs.

However, it is **highly recommended to ultimately set them via the software API**. This way, in the event a device is replaced, you can rely on your software to properly configured the new device, without having to remember to use Tuner to apply the correct values.

A general recommendation is to:

- Configure all devices during robot-bootup using the API,
- Use Tuner to dial values quickly during testing/calibration.
- Export the settings so they are not lost.
- Update your software config values so that Tuner is no longer necessary.

18.2.4 Control Signals

The majority of the behavior in the Talon/Victor is controlled via configs, however there is a small number of control signals that are controlled via the API.

This list includes:

- Current Limit **Enable** (though the thresholds are configs)
- Voltage Compensation **Enable** (though the nominal voltage is a config)
- Control Mode and Target/Output demand (percent, position, velocity, etc.)
- Invert direction and sensor phase
- Closed-loop slot selection [0,3] for primary and aux PID loops.
- Neutral mode override (convenient to temporarily override configs)
- Limit switch override (convenient to temporarily override configs)
- Soft Limit override (convenient to temporarily override configs)
- Status Frame Periods

These control signals do not require periodic calls to ensure they “stick”. All of the above signals are automatically restored even after motor controller is power cycled during use except for Status Frame Periods, which can be manually restore by polling for device resets via `hasResetOccurred()`.

Note: WPI motor safety features may require periodic calls to `Set()` if team software has chosen to enable it.

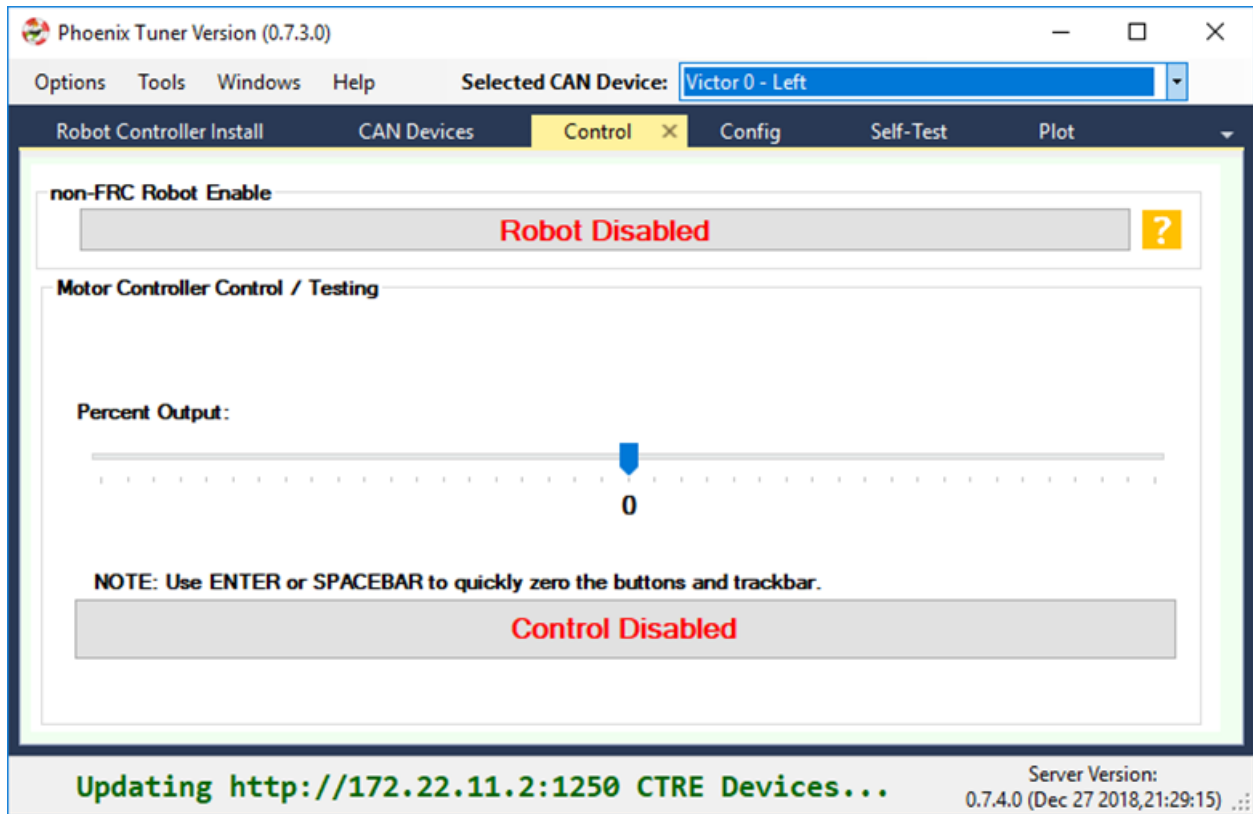
Note: The override control signals are useful for applications that require temporarily disabling or changing behavior. For example, overriding-disable the soft limits while performing a self-calibration routine to tare sensors, then restoring soft limits for robot operation.

Note: The routines to manipulate control signals are not prefixed with `config*` to highlight that they are not configs

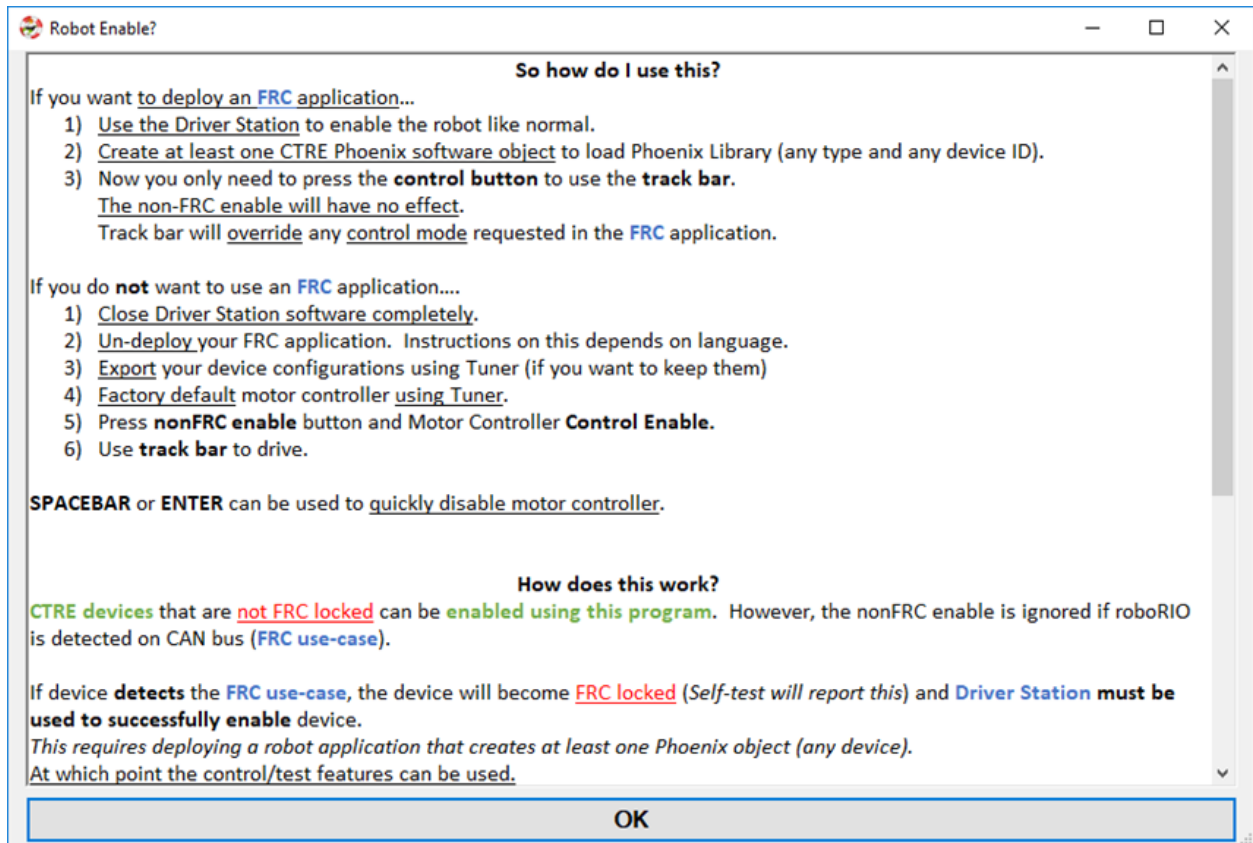
18.3 Test Drive with Tuner

Navigate to the control tab to view the control interface. Notice there are two enable/disable buttons. One is for non-FRC style robot-enable (alternative to the Driver Station enable), and one is for Motor Controller Control-Enable.

Press on the question mark next to the robot disabled/enabled button.



This will reveal the full explanation of how to safely enable your motor controller. Follow the appropriate instructions depending on if you want to use Driver Station for your robot-enable.



18.3.1 Setting up non-FRC Control

In order to enable without the Driver Station you must use a non-roboRIO platform and disconnect the roboRIO from the CAN bus.

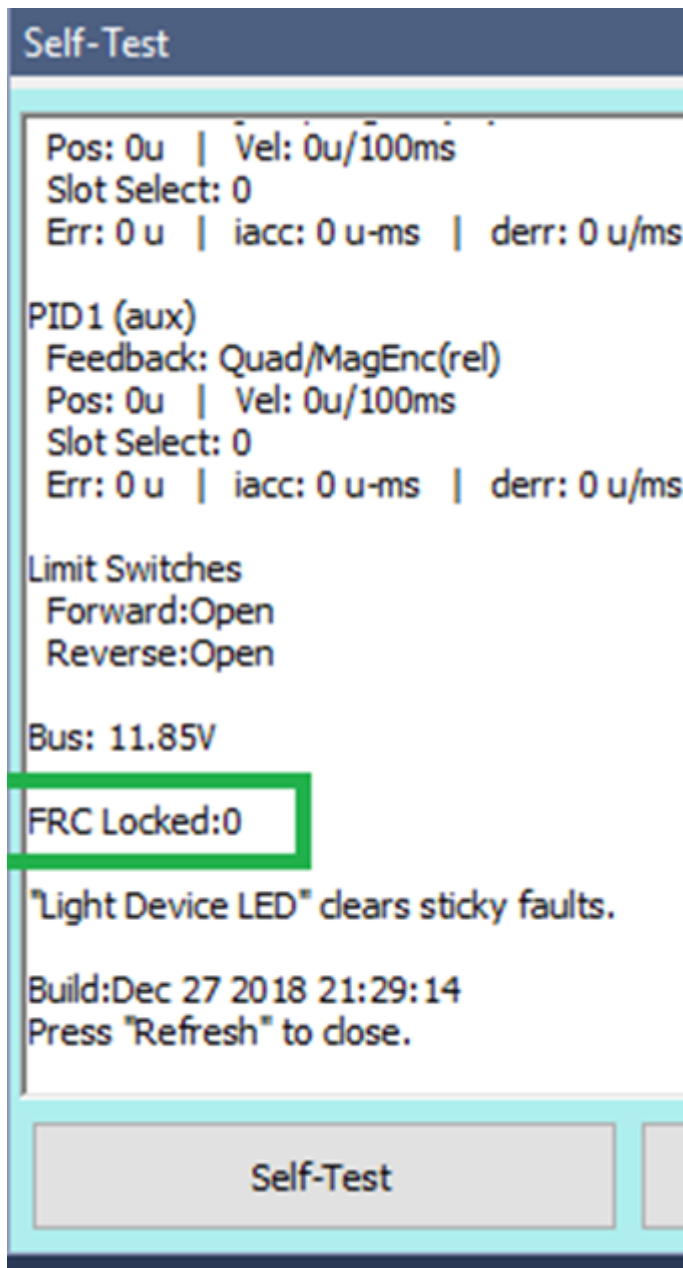
With an FRC roboRIO, you must *always* use the Driver Station to enable.

18.3.2 Confirm FRC Unlock

Self-test Snapshot Motor Controller to confirm device FRCLocked = 0.

If device is FRC Locked (=1), use factory default in the config tab to clear the state. Note that if an FRC roboRIO is on the CAN bus, the motor controller will immediately FRC Lock again.

Note: Use the config export tool if you need to keep your config settings.



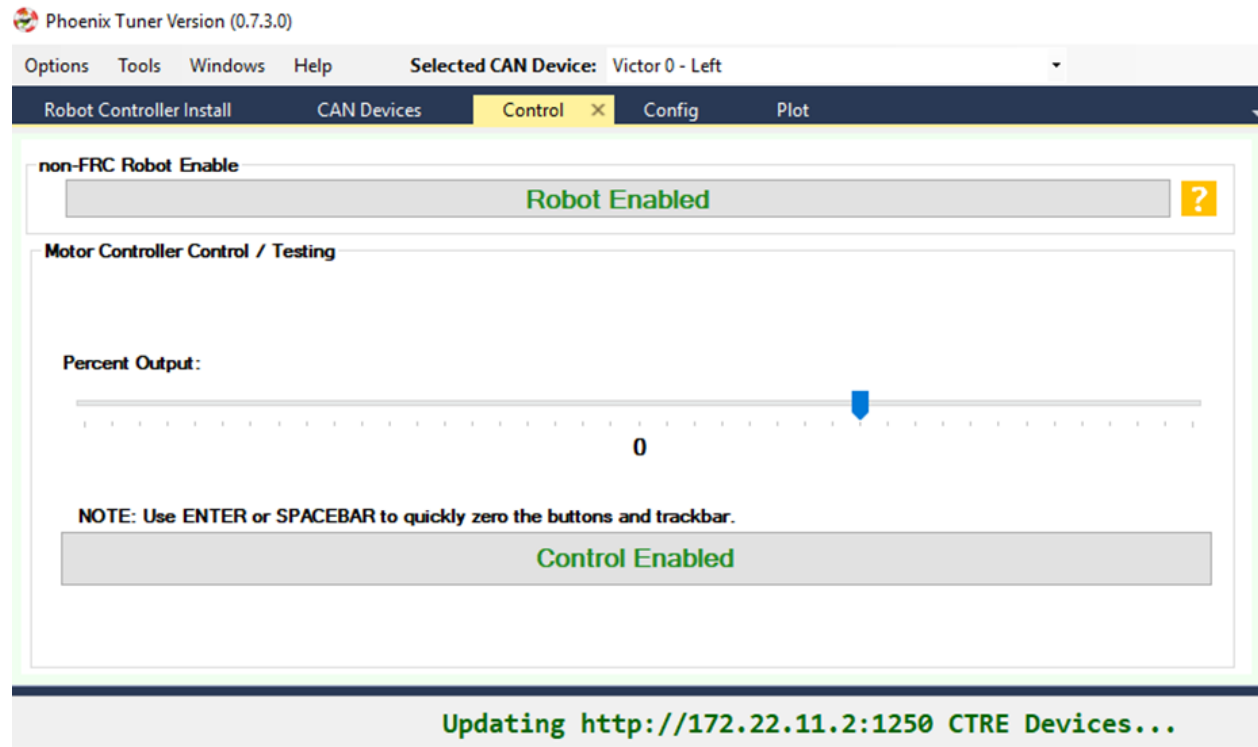
18.3.3 Control tab

Press both Robot Enabled and Control Enabled. At this point you can use the track bar to drive the Victor/Talon.

Note: If you do connect with a roboRIO, the Talon/Victor will FRC Lock again. At which point you must use the driver station to enable, and you no longer need to use the non-FRC Robot enable in Tuner.

Note: Spacebar or enter can be used to clear the control tab and neutral the selected motor

controller.



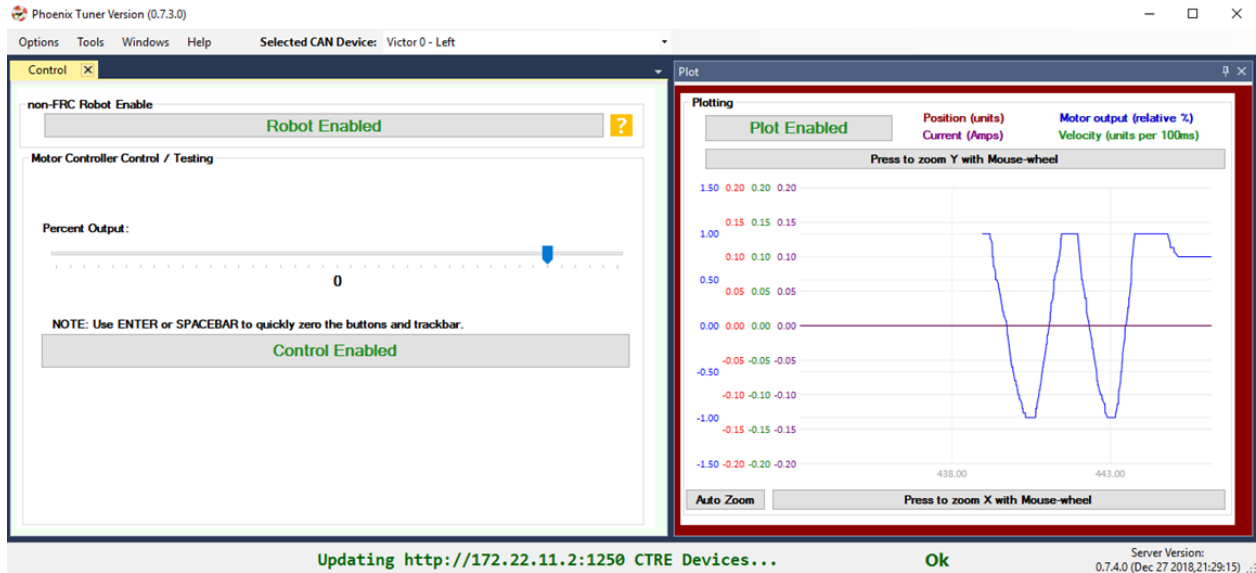
18.3.4 Plot tab

Now open the Plot window. Drive the motor controller while observing the plot. Confirm the blue motor output curve matches LED behavior and trackbar. Confirm motor movement follows expectations.

Note: Press the Plot enable button to effectively pause the plot for review

Note: Use the Zoom buttons to select whether the mouse adjust the Y or X axis.

Note: If using a Victor SPX, current-draw will always read zero (SPX does not have current-measurement features).



Tip: Plot can be used anytime, regardless of what is commanding the motor controller (FRC or non-FRC).

18.4 Test Drive with Robot Controller

Next we will create control software in the roboRIO. Currently this is necessary for more advanced control. This is also required for controlling your robot during competition.

Tip: The latest version of Tuner allows for testing most closed-loop control modes without writing software.

18.4.1 Java: Sample driving code

Below is a simple example that reads the Joystick and drives the Talon

```
package frc.robot;

import com.ctre.phoenix.motorcontrol.ControlMode;
import com.ctre.phoenix.motorcontrol.can.TalonSRX;

import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.TimedRobot;

public class Robot extends TimedRobot {
    TalonSRX _talon0 = new TalonSRX(0); // Change '0' to match device ID in Tuner. Use ↵
    ↵ VictorSPX for Victor SPXs
    Joystick _joystick = new Joystick(0);

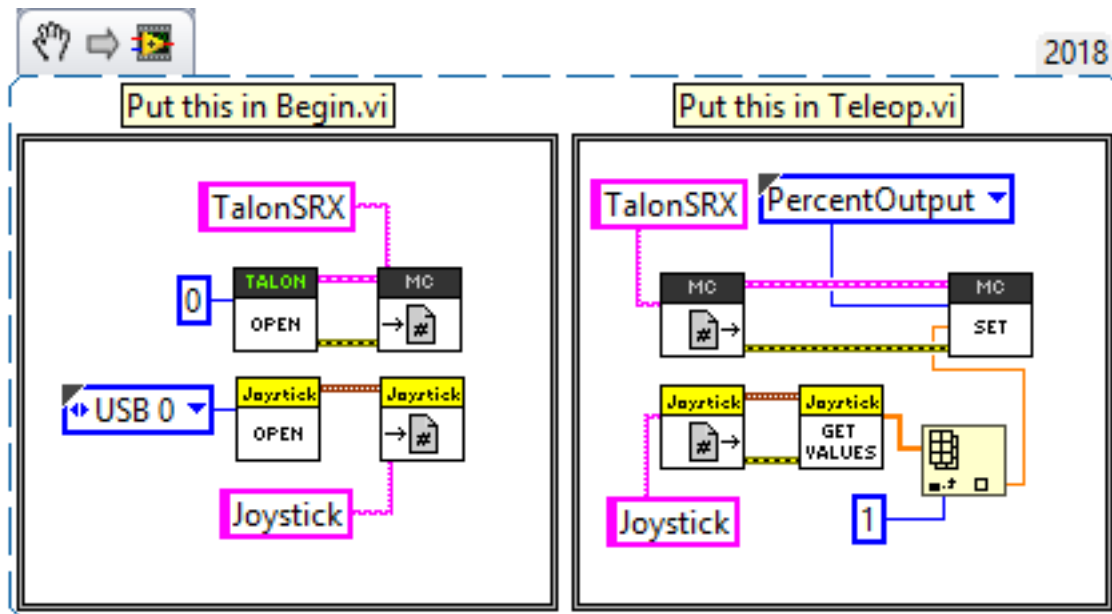
    @Override
```

(continues on next page)

(continued from previous page)

```
public void teleopPeriodic() {  
    double stick = _joystick.getRawAxis(1);  
    _talon0.set(ControlMode.PercentOutput, stick);  
}  
}
```

Tip: Image below can be dragged/dropped into LabVIEW editor.



Deploy the project, and confirm success.

Note: WPI's terminal output may read "Build" successful despite the project was deployed.

```

8  package frc.robot;
9
10 import com.ctre.phoenix.motorcontrol.ControlMode;
11 import com.ctre.phoenix.motorcontrol.can.TalonSRX;
12
13 import edu.wpi.first.wpilibj.Joystick;
14 import edu.wpi.first.wpilibj.TimedRobot;
15
16 public class Robot extends TimedRobot {
17     TalonSRX _talon0 = new TalonSRX(0);
18     Joystick _joystick = new Joystick(0);
19
20     @Override
21     public void teleopPeriodic() {
22         double stick = _joystick.getRawAxis(0);
23         _talon0.set(ControlMode.PercentOutput, stick);
24     }
25 }
26

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL**

```

-C-> chmod +x "/home/lvuser/MyJavaProject-1.jar"; chown lvuser "/home/lvuser/MyJavaProject-1.jar" @ /
-C-> sync @ /home/lvuser
-[-1]->
-C-> . /etc/profile.d/natinst-path.sh; /usr/local/frc/bin/frcKillRobot.sh -t -r 2> /dev/null @ /home/

```

> Task :deployNativeZipRoborio

42 file(s) are up-to-date and were not deployed

```
-C-> chmod -R 777 "/usr/local/frc/third-party/lib" || true; chown -R lvuser:ni "/usr/local/frc/third-
```

```
-C-> ldconfig @ /usr/local/frc/third-party/lib
```

Deprecated Gradle features were used in this build, making it incompatible with Gradle 6.0.

Use '--warning-mode all' to show the individual deprecation warnings.

See https://docs.gradle.org/5.0/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 5s

10 actionable tasks: 8 executed, 2 up-to-date

Note: Before you enable the DS, spin the Joystick axis so it reaches the X and Y extremities are reached. USB Gamepads calibrate on-the-fly so if the Gamepad was just inserted into the DS, it likely has not auto detected the max mechanical range of the sticks.

Note: Make sure joystick is detected by the DS before enabling.

Note: getRawAxis may not return a positive value on forward-stick. Confirm this by watching Talon/Victor LED. Green suggests a positive output.

Enable the Driver Station and confirm:

- motor drive in both directions using gamepad stick.

- motor controller LEDs show green for forward and red for reverse

Disable Driver Station after finished testing.

Note: If the LED is solid orange than use Tuner to determine the cause. Self-test Snapshot will report the current state of the motor controller (do this while troubleshooting). Confirm firmware is up to date.

18.5 Open-Loop Features

After some rudimentary testing, you will likely need to configure several open-loop features of the Talon SRX and Victor SPX.

Note: We recommend configuring Inverts and Followers first.

18.5.1 Inverts

To determine the desired invert of our motor controller, we will add two more lines of call. `SetInverted` is added to decide if motor should spin clockwise or counter clockwise when told to move positive/forward (green LEDs).

We also multiply the joystick so that forward is positive (intuitive). This can be verified by watching the console print in the Driver Station.

```
package frc.robot;
import com.ctre.phoenix.motorcontrol.*;
import com.ctre.phoenix.motorcontrol.can.*;

import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.TimedRobot;

public class Robot extends TimedRobot {
    TalonSRX _talon0 = new TalonSRX(0);
    Joystick _joystick = new Joystick(0);

    @Override
    public void teleopInit() {
        _talon0.setInverted(false); // pick CW versus CCW when motor controller is
        // positive/green
    }

    @Override
    public void teleopPeriodic() {
        double stick = _joystick.getRawAxis(1) * -1; // make forward stick positive
        System.out.println("stick:" + stick);

        _talon0.set(ControlMode.PercentOutput, stick);
    }
}
```

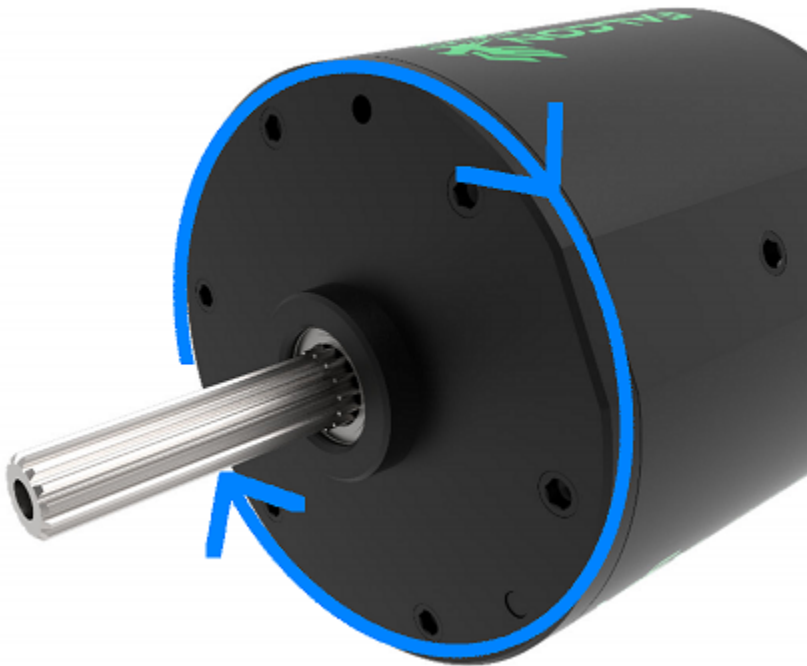
Tip: Image below can be dragged/dropped into LabVIEW editor.



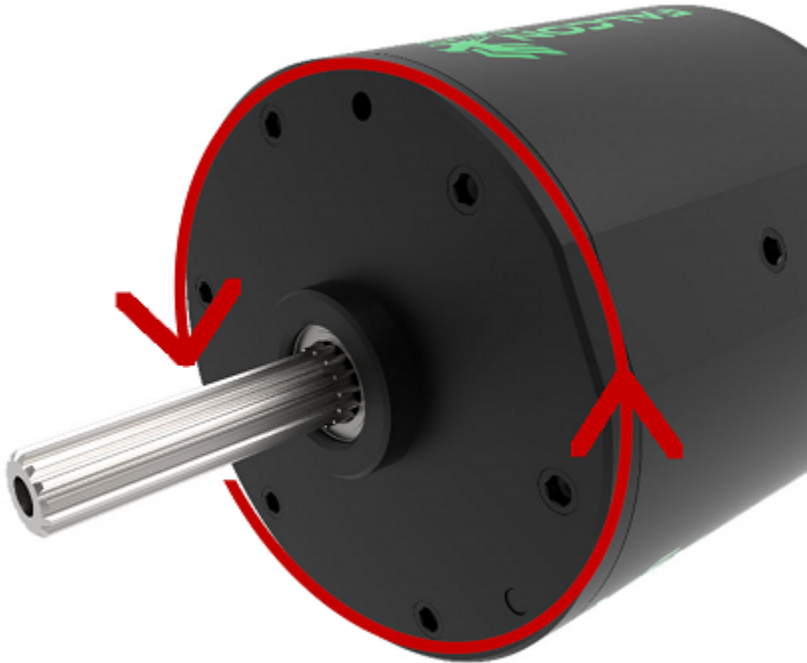
Talon FX Specific Inverts

Talon FX has a new set of invert types that are specific to it, *TalonFXInvertType.Clockwise* and *TalonFXInvertType.CounterClockwise*. These new invert types allow the user to know exactly what direction the Falcon 500 will spin. These invert types are from the perspective of looking at the face of the motor.

Below is an image demonstrating the Falcon's **Clockwise** rotation:



And below is the Falcon's **CounterClockwise** rotation:



18.5.2 Follower

If a mechanism requires multiple motors, then there are likely multiple motor controllers. The Follower feature of the Talon FX/SRX and Victor SPX is a convenient method to keep two or more motor controller outputs consistent. If you have an external sensor for closed-looping, connect that to the “master” Talon SRX (unless it is a remote sensor such as CAN-coder/CANifier/Pigeon).

Below we’ve added a new Victor to follow Talon 0.

Generally, a follower is intended to match the direction of the master, or drive in the opposite direction depending on mechanical orientation. In previous seasons teams would have to update the bool true/false of the follower to match or oppose the master manually.

Starting in 2019, C++/Java users can set the `setInverted(InvertType)` to instruct the motor controller to either match or oppose the direction of the master instead.

```
package frc.robot;

import com.ctre.phoenix.motorcontrol.*;
import com.ctre.phoenix.motorcontrol.can.*;

import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.TimedRobot;

public class Robot extends TimedRobot {
    TalonSRX _talon0 = new TalonSRX(0);
    VictorSPX _victor0 = new VictorSPX(0);
    Joystick _joystick = new Joystick(0);

    @Override
    public void teleopInit() {
```

(continues on next page)

(continued from previous page)

```

_victor0.follow(_talon0);

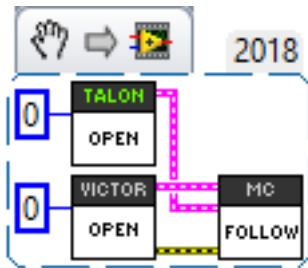
_talon0.setInverted(false); // pick CW versus CCW when motor controller is
↪ positive/green
_victor0.setInverted(InvertType.FollowMaster); // match whatever talon0 is
//_victor0.setInverted(InvertType.OpposeMaster); // opposite whatever talon0 is
}

@Override
public void teleopPeriodic() {
    double stick = _joystick.getRawAxis(1) * -1; // make forward stick positive
    System.out.println("stick:" + stick);

    _talon0.set(ControlMode.PercentOutput, stick);
}
}

```

Tip: Image below can be dragged/dropped into LabVIEW editor.



Note: LabVIEW does not support using InvertType to follow master or oppose master

Enable the Driver Station and slowly drive both MCs from neutral. Confirm both LEDs are blinking the same color.

Disable Driver Station when complete.

To confirm motor controllers are truly driving in the same direction, disconnect the master motor controller from its motor.

Enable the Driver Station and confirm follower motor direction matches previously measured master motor direction.

Disable Driver Station when complete.

Open Tuner and select the master motor controller.

Open plot tab and enable plotter while driving motor controller

Confirm current plot is appropriate. If motors are free-spinning, then current should be near 0 if motor output is constant. When testing drive train, the robot should be rested on a crate/tote to ensure all wheels spin freely.

Select follower motor in Tuner, and confirm current via plot.

Note: Follower mode can be canceled by calling set() with any other control mode, or calling

neutralOutput().

Note: Calling follow() in the periodic loop is not required, but also does not affect anything in a negative way.

Controlling Followers with Phoenix Tuner

Oftentimes you want to test/tune a mechanism with a master motor controller and one or more followers. This can be accomplished with Phoenix Tuner in the same manner as if there was only one controller, **as long as the followers are configured to follow the master**. This means you **cannot** run a temporary diagnostic server to control multiple motor controllers at the same time.

It is imperative to make sure the followers are configured correctly by **following the steps above**. The followers will use their settings from the user application, even when following a master controlled by Tuner.

Tip: This is the recommended way to tune two or more mechanically linked motors. By having one motor controller as a master, it will handle the PID closed looping while all followers match the applied output of the master.

18.5.3 Neutral Mode

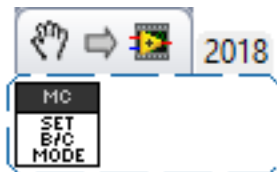
You may note that when the motor output transitions to neutral, the motors free spin (coast) in the last direction they were driven. If the Talon/Victor is set to “coast” neutral mode, then this is expected. The neutral mode can also be set to “brake” to electrically common the motor leads during neutral, causing a deceleration that combats the spinning motor motion.

Tip: You can use Talon FX’s ConfigStatorCurrentLimit method to dial in how strong the brake is.

Note: SetNeutralMode() can be used change the neutral mode on the fly.

```
TalonSRX talon = new TalonSRX(0);  
talon.setNeutralMode(NeutralMode.Brake);
```

Tip: Image below can be dragged/dropped into LabVIEW editor.



Follower motor controllers have separate neutral modes than their masters, so you must choose both. Additionally, you may want to mix your neutral modes to achieve a partial electric brake when using multiple motors.

18.5.4 Neutral Deadband

A device's neutral deadband is the region where the controller demotes its output to neutral. This can be configured in your robot code, with a default value of 0.04 or 4%, and a range of [0.001, 0.25] or [0.1%, 25%].

```
_talon.configNeutralDeadband(0.001); /* Configures _talon to use a neutral deadband of 0.1% */
```

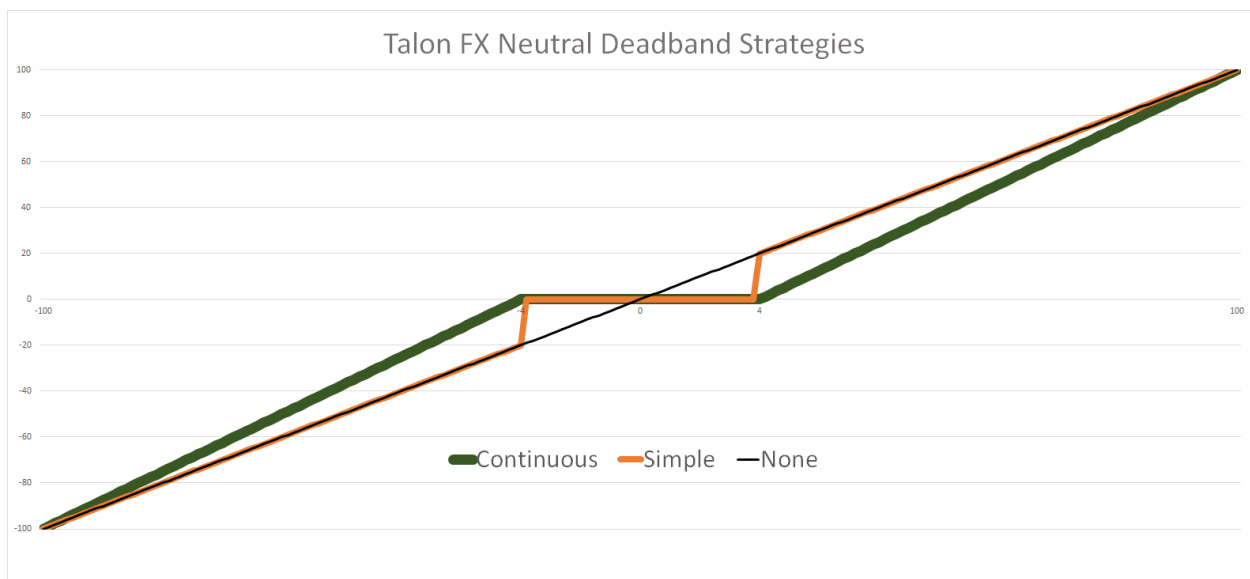
Talon FX has 3 different deadband strategies based on its state. They are *Simple*, *Continuous*, and *None*.

A *Simple* deadband will demote any requested output within the region to neutral, and otherwise uphold the requested demand. An example of this is with a configured deadband of 4% and a requested output of 4% will be 0%, 5% output will be 5%, and 100% will be 100%. This is used in the majority of circumstances so it's obvious that the requested output is the applied output outside the neutral deadband.

A *Continuous* deadband is similar to a simple deadband in that it demotes any requested output within the region to neutral, but outside the region it will scale the applied output so it's continuous out of the deadband thresholds. This allows for a smooth transition out of neutral. With a 4% deadband, a requested output of 4% will result in an applied output of 0%, requesting 5% will bring it to 1%, and 100% will be 100%.

A *None* deadband will not uphold the deadband whatsoever. A deadband of 4% with 4% requested output will apply 4%, 5% is 5%, and 100% is 100%. This is used only in follower mode so you don't have to configure the deadband of your followers, only of the master.

The below graph highlights this, exaggerating the effect to make it obvious.



The below table details what neutral deadband strategy the Talon FX uses under the various states.

Table 1: Talon FX Neutral Deadband Strategies

Mode	Condition	Deadband Type
PWM Control	X	Continuous
Percent Output	Voltage Compensation Disabled	Continuous
Percent Output	Voltage Compensation Enabled	Simple
Closed Loop	X	Simple
Auxiliary Follower	X	Simple
Follower	X	None

18.5.5 Ramping

The motor controller can be set to honor a ramp rate to prevent instantaneous changes in throttle. This ramp rate is in effect regardless of which mode is selected (throttle, slave, or closed-loop).

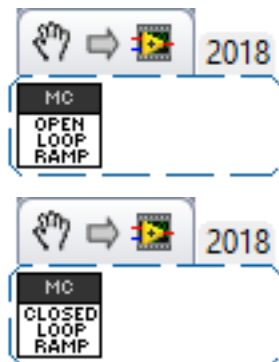
Ramp can be set in time from neutral to full using `configOpenLoopRampRate()`.

Note: `configClosedLoopRampRate()` can be used to select the ramp during closed-loop (sensor) operations.

Note: The slowest ramp possible is ten seconds (from neutral to full), though this is quite excessive.

```
TalonSRX talon = new TalonSRX(0);
talon.configOpenloopRamp(0.5); // 0.5 seconds from neutral to full output (during
↪open-loop control)
talon.configClosedloopRamp(0); // 0 disables ramping (during closed-loop control)
```

Tip: Images below can be dragged/dropped into LabVIEW editor.



18.5.6 Peak/Nominal Outputs

Often a mechanism may not require full motor output. The application can cap the output via the peak forward and reverse config setting (through Tuner or API).

Additionally, the nominal outputs can be selected to ensure that any non-zero requested motor output gets promoted to a minimum output. For example, if the nominal forward is set to +0.10 (+10%), then any motor request within (0%, +10%) will be promoted to +10% assuming request is beyond the neutral dead band. This is useful for mechanisms that require a minimum output for movement, and can be used as a simpler alternative to the KI (integral) component of closed-looping in some circumstances.

18.5.7 Voltage Compensation

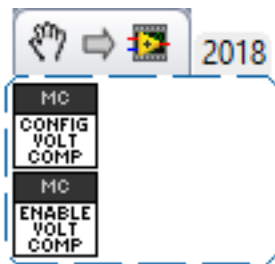
Talon FX/SRX and Victor SPX can be configured to adjust their outputs in response to the battery voltage measurement (in all control modes). Use the voltage compensation saturation config to determine what voltage represents 100% output.

Then enable the voltage compensation using `enableVoltageCompensation()`.

Advanced users can adjust the Voltage Measurement Filter to make the compensation more or less responsive by increasing or decreasing the filter. This is available via API and via Tuner

```
TalonSRX talon = new TalonSRX(0);
talon.configVoltageCompSaturation(11); // "full output" will now scale to 11 Volts
↳ for all control modes when enabled.
talon.enableVoltageCompensation(true); // turn on/off feature
```

Tip: Image below can be dragged/dropped into LabVIEW editor.



18.5.8 Current Limit

Legacy API

Talon FX/SRX supports current limiting in all control modes.

The limiting is characterized by three configs:

- Peak Current (Amperes), threshold that must be exceeded before limiting occurs.
- Peak Time (milliseconds), thresholds that must be exceed before limiting occurs
- Continuous Current (Amperes), maximum allowable current after limiting occurs.

```
TalonSRX talon = new TalonSRX(0);
talon.configPeakCurrentLimit(30); // don't activate current limit until current
↳exceeds 30 A ...
talon.configPeakCurrentDuration(100); // ... for at least 100 ms
talon.configContinuousCurrentLimit(20); // once current-limiting is activated, hold at
↳20A
talon.enableCurrentLimit(true);
```

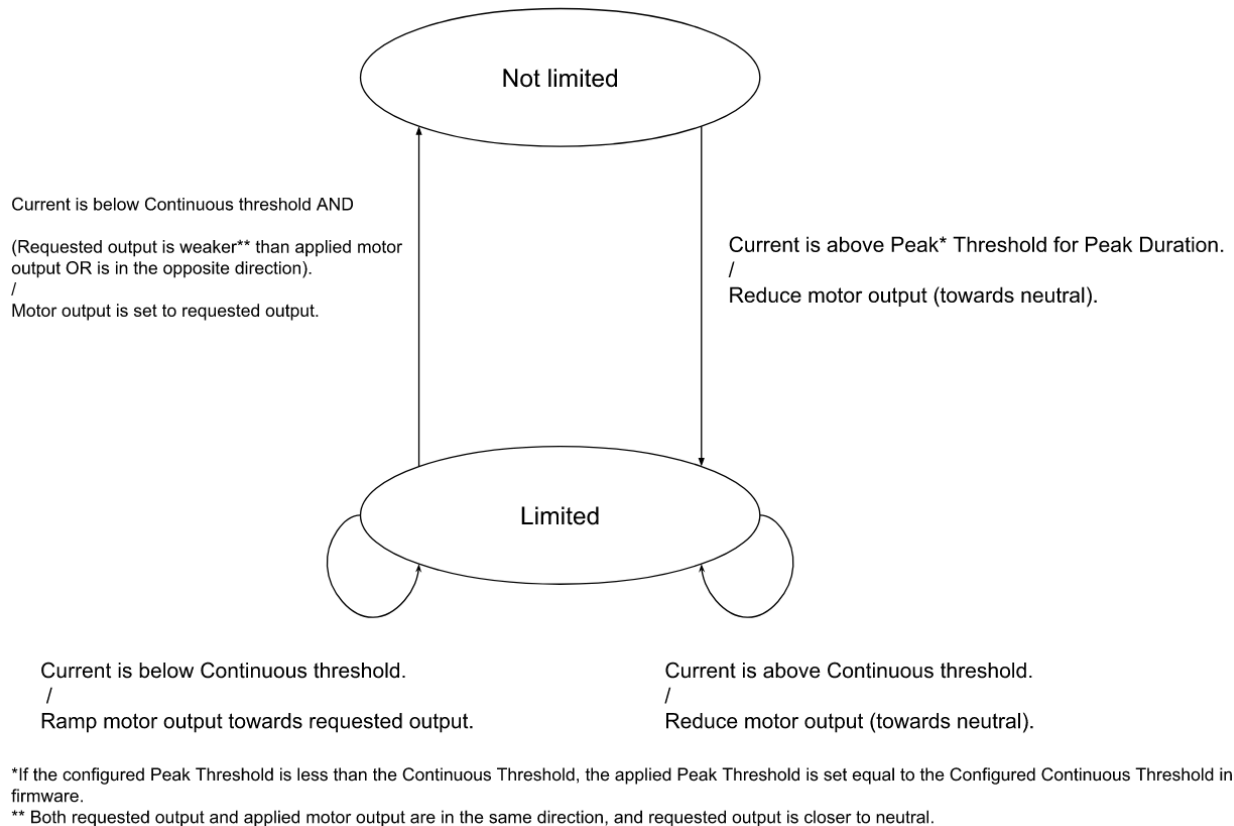
Tip: Image below can be dragged/dropped into LabVIEW editor.



If enabled, Talon SRX will monitor the supply-current looking for a conditions where current has exceeded the Peak Current for at least Peak Time. If detected, output is reduced until current measurement is at or under Continuous Current.

Note: If Peak current limit is set less than continuous limit, peak current limit will be set equal to continuous current limit.

Once limiting is active, current limiting will deactivate if motor controller can apply the requested motor output and still measure current-draw under the Continuous Current Limit.



After setting the three configurations, current limiting must be enabled via `enableCurrentLimit()` or LabVIEW VI.

Note: Use Self-test Snapshot to confirm if Current Limiting is occurring

Note: If peak limit is less than continuous limit, peak is set equal to continuous

Note: If you only want continuous limiting, you should set peak limit to 0

New API in 2020

Talon FX supports both stator(output) current limiting and supply(input) current limiting.

Supply current is current that's being drawn at the input bus voltage. Stator current is current that's being drawn by the motor.

Supply limiting (supported by Talon SRX and FX) is useful for preventing breakers from tripping in the PDP.

Stator limiting (supported by Talon FX) is useful for limiting acceleration/heat.

The new API leverages the `configSupplyCurrentLimit` and `configStatorCurrentLimit` routines. The configs are similar to the existing legacy API, but the configs have been renamed to better

communicate the design intent. For example, instead of `configPeakCurrentLimit`, the setting is referred to as `triggerThresholdCurrent`.

```
/**
 * Configure the current limits that will be used
 * Stator Current is the current that passes through the motor stators.
 * Use stator current limits to limit rotor acceleration/heat production
 * Supply Current is the current that passes into the controller from the supply
 * Use supply current limits to prevent breakers from tripping
 *
 *
 *                                     enabled |
↪ Limit(amp) | Trigger Threshold(amp) | Trigger Threshold Time(s) */
_tal.configStatorCurrentLimit(new StatorCurrentLimitConfiguration(true,    20,
↪      25,                      1.0));
_tal.configSupplyCurrentLimit(new SupplyCurrentLimitConfiguration(true,    10,
↪      15,                      0.5));
```

An example of this is available on our [Github Examples](#) repository

18.6 Reading status signals

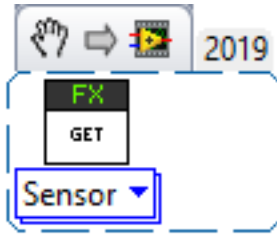
The Talon FX/SRX and Victor SPX transmit most of their status signals periodically, i.e. in an unsolicited fashion. This improves bus efficiency by removing the need for “request” frames, and guarantees that the signals necessary for the wide range of use cases they support are available.

These signals are available in API regardless of what control mode the Talon SRX is in. Additionally the signals can be polled using Phoenix Tuner using the Self-test Snapshot button.

Included in the list of signals are:

- Quadrature Encoder Position, Velocity, Index Rise Count, Pin States (A, B, Index)
- Analog-In Position, Analog-In Velocity, 10bit ADC Value,
- Battery Voltage, Current, Temperature
- Fault states, sticky fault states,
- Limit switch pin states
- Applied Throttle (duty cycle) regardless of control mode.
- Applied Control mode: Voltage % (duty-cycle), Position/Velocity closed-loop, or slave follower.
- Brake State (coast vs brake)
- Closed-Loop Error, the difference between closed-loop set point and actual position/velocity.
- Sensor Position and Velocity, the signed output of the selected Feedback device (robot must select a Feedback device, or rely on default setting of Quadrature Encoder).
- Integrated Sensor (Talon FX).
- Magnet position and strength (CANCoder).

Tip: In LabVIEW, these signals can all be obtained from the “Get” VI from the motor controller’s sub-palette. Choose the type of signals desired from the VI’s drop-down menu.



18.7 Limit Switches

Talon SRX and Victor SPX have limit features that will auto-neutral the motor output if a limit switch activates. **Talon SRX** in particular can automatically do this **when limit switches are connected via the Gadgeteer feedback port**.

An “out of the box” Talon will **default with the limit switch setting of “Normally Open”** for both forward and reverse. This means that motor drive is allowed when a limit switch input is not closed (i.e. not connected to ground). When a limit switch input is closed (is connected to ground) the Talon SRX will disable motor drive and individually blink both LEDs red in the direction of the fault (red blink pattern will move towards the M+/white wire for positive limit fault, and towards M-/green wire for negative limit fault).

Since an “out of the box” Talon will likely not be connected to limit switches (at least not initially) and because limit switch inputs are internally pulled high (i.e. the switch is open), the limit switch feature is default to “normally open”. This ensures an “out of the box” Talon will drive even if no limit switches are connected.

For more information on Limit Switch wiring/setup, see the Talon SRX User’s Guide.

Forward Limit Switch Mode	Limit Switch NO pin	Limit Switch NC pin	Limit Switch COM pin	Motor Drive Switch open Fwd. throttle	Motor Drive Switch closed Fwd. throttle	*Voltage (Switch Open)	*Voltage (Switch Closed)
Normally Open	pin4	N.A.	pin10	Y	N	~2.5V	0 V
Normally Closed	N.A.	pin4	pin10	N	Y	0 V	~2.5V
Disabled	N.A.	N.A.	N.A.	Y	Y	N.A.	N.A.
Reverse Limit Switch Mode	Limit Switch NO pin	Limit Switch NC pin	Limit Switch COM pin	Motor Drive Switch open Rev. throttle	Motor Drive Switch closed Rev. throttle	*Voltage (Switch Open)	*Voltage (Switch Closed)
Normally Open	pin8	N.A.	pin10	Y	N	~2.5V	0 V
Normally Closed	N.A.	pin8	pin10	N	Y	0 V	~2.5V
Disabled	N.A.	N.A.	N.A.	Y	Y	N.A.	N.A.
*Measured voltage at the Talon SRX Limit Switch Input pin.							
Limit Switch Input Forward Input - pin4 on Talon SRX							
Limit Switch Input Reverse Input - pin8 on Talon SRX							
Limit Switch Ground - pin10 on Talon SRX							

Limit switch features can be disabled or changed to “Normally Closed” in Tuner and in API.

Note: When the source is set to Gadgeteer, the “Device ID” field is ignored. This config is used for **remote limit switches** (see next section).

Confirm the limit switches are functional by applying a **weak positive motor output** while tripping the forward limit switch.

Note: The motor does not have to be physically connected to the motor-controller if tester can artificially assert physical limit switch.

```

/* Configured forward and reverse limit switch of Talon to be from a feedback
connector and be normally open */
Hardware.leftTalonMaster.configForwardLimitSwitchSource(LimitSwitchSource.
FeedbackConnector, LimitSwitchNormal.NormallyOpen, 0);
Hardware.leftTalonMaster.configReverseLimitSwitchSource(LimitSwitchSource.
FeedbackConnector, LimitSwitchNormal.NormallyOpen, 0);

```


18.7.1 Limit Switch Override Enable

The enable state of the limit switches can be overridden in software. This can be called at any time to enable or disable both limit switches.

Generally you should call this instead of a config if you want to dynamically change whether you are using the limit switch or not inside a loop. This value is not persistent across power cycles.

```
/* Limit switches are forced disabled on Talon and forced enabled on Victor */
Hardware.leftTalonMaster.overrideLimitSwitchesEnable(false);
Hardware.rightVictorMaster.overrideLimitSwitchesEnable(true);;
```

18.7.2 Limit Switch As Digital Inputs

Limit switches can also be treated as digital inputs. This is done in Java/C++ by using the `isFwdLimitSwitchClosed` & `isRevLimitSwitchClosed` method.

```
_talon.getSensorCollection().isFwdLimitSwitchClosed();
_talon.getSensorCollection().isRevLimitSwitchClosed();
```

Note: The sensor being closed returns true in all cases, and the sensor being open returns false in all cases, regardless of normally open/normally closed setting. This ensures there is no ambiguity in the function name.

18.7.3 Remote Limit Switches

A Talon SRX or Victor SPX can use a remote sensor as the limit switch (such as another Talon SRX or CANifier).

Change the Limit Forward/Reverse Source to Remote Talon or Remote CANifier. Then config the Limit Forward/Reverse Device ID for the remote Talon or CANifier.

```
/* Configured forward and reverse limit switch of a Victor to be from a Remote Talon,
↳SRX with the ID of 3 and normally closed */
Hardware.rightVictorMaster.configForwardLimitSwitchSource(RemoteLimitSwitchSource.
↳RemoteTalonSRX, LimitSwitchNormal.NormallyClosed, 3, 0);
Hardware.rightVictorMaster.configReverseLimitSwitchSource(RemoteLimitSwitchSource.
↳RemoteTalonSRX, LimitSwitchNormal.NormallyClosed, 3, 0);
```

Use Self-test Snapshot on the motor-driving motor controller to confirm limit switches are interpreted correctly. If they are not correct, then Self-test Snapshot the remote device to determine the issue.

18.8 Soft Limits

Soft limits can be used to disable motor drive when the “Sensor Position” is outside of a specified range. Forward throttle will be disabled if the “Sensor Position” is greater than the Forward Soft Limit. Reverse throttle will be disabled if the “Sensor Position” is less than the

Reverse Soft Limit. The respective Soft Limit Enable must be enabled for this feature to take effect.

```
/* Talon configured to have soft limits 10000 native units in either direction and  
↔enabled */  
rightMaster.configForwardSoftLimitThreshold(10000, 0);  
rightMaster.configReverseSoftLimitThreshold(-10000, 0);  
rightMaster.configForwardSoftLimitEnable(true, 0);  
rightMaster.configReverseSoftLimitEnable(true, 0);
```

The settings can be set and confirmed in Phoenix Tuner

Troubleshooting and Frequently Asked Questions

19.1 Driver Station Messages

19.1.1 What do I do when I see errors in Driver Station?

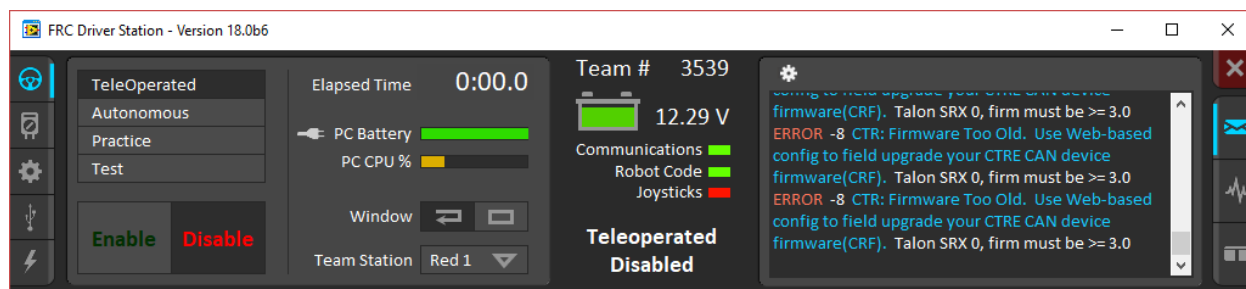
DS Errors should be addressed as soon as they appear. This is because:

- Phoenix API will report if a device is missing, not functioning, has too-old firmware, etc.
- If errors are numerous and typical, then users cannot determine if there is a new problem to address.
- A large stream of errors can bog down the DriverStation/roboRIO. Phoenix Framework has a debouncing strategy to ensure this does not happen, but not all libraries do this.

Phoenix DS errors occur on call. Meaning VIs/API functions must be called in robot code for any errors to occur. When an error does occur, a stack trace will report where in the robot code to look.

The Debouncing Strategy that Phoenix uses is 3 seconds long. Phoenix keys a new error on device ID & function. This is to ensure that all unique errors are logged while making sure the DriverStation/roboRIO does not generate excessive errors.

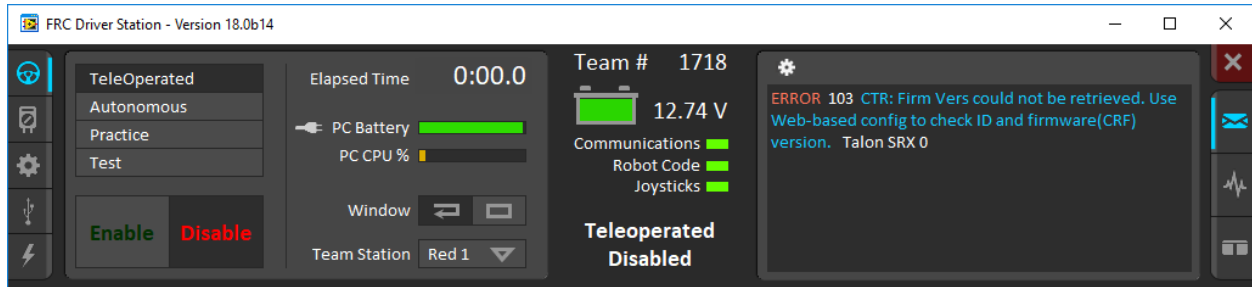
19.1.2 Driver Station says the firmware is too old.



Use Phoenix Tuner to update the firmware of the device.

Note that the robot application must be restarted for the firmware version check to clear. This can be done by redeploying the robot application or simply restarting the robot.

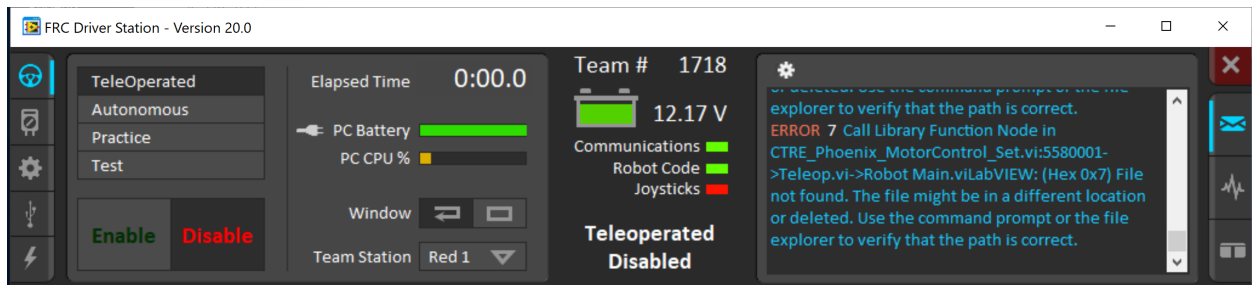
19.1.3 Driver Station says the firmware could not be retrieved and to check the firmware and ID.



This usually indicates that your **device ID is wrong** in your robot software, or your firmware is **very old**.

Use Phoenix Tuner to check your device IDs and make sure your firmware is up-to-date.

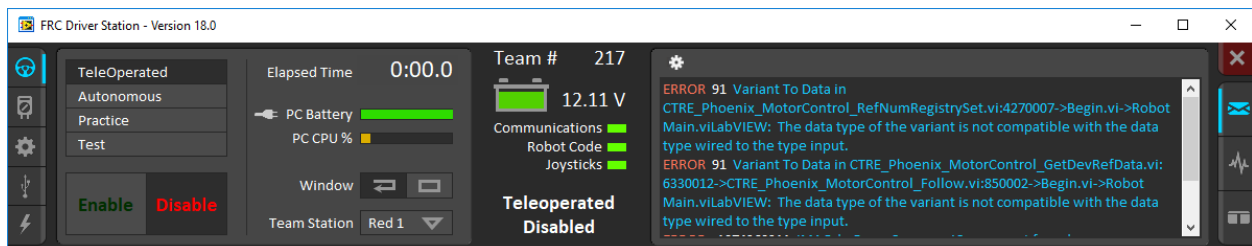
19.1.4 Driver Station Says “ERROR 7 Call Library Function Node...”



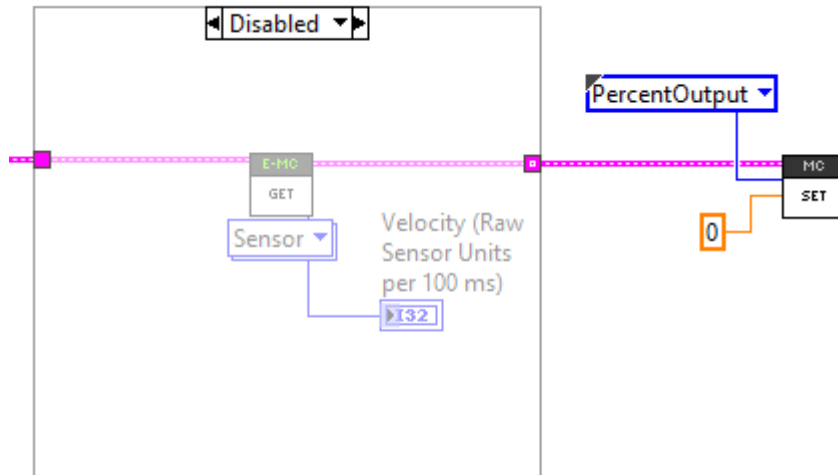
This can be seen when the Phoenix libraries are not present on the roboRIO.

This can be fixed by following the process to prepare the roboRIO for *LabVIEW*.

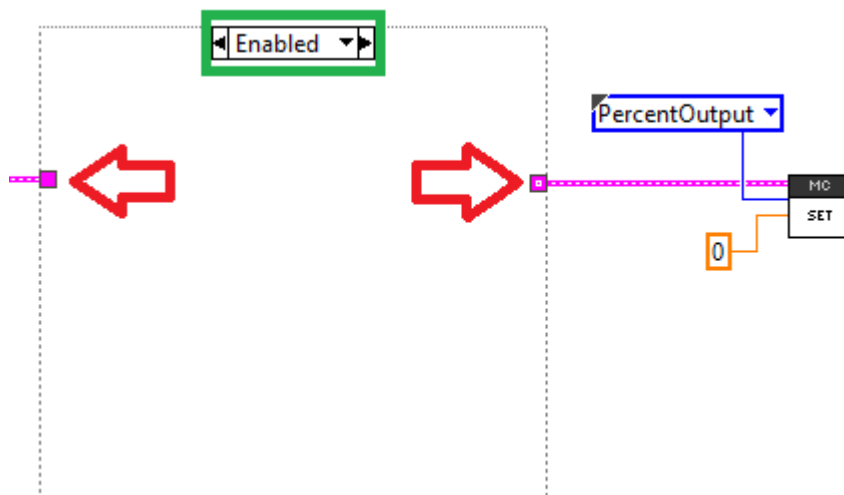
19.1.5 Driver Station Says Variant To Data in ...

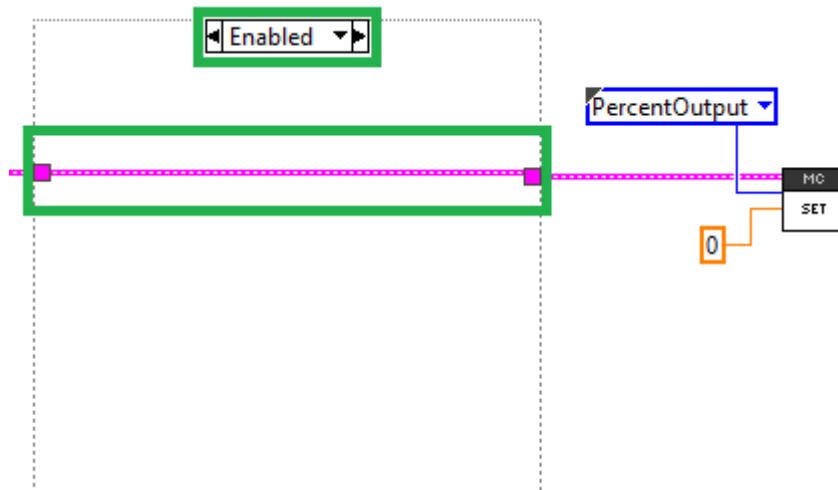


This is usually caused by a diagram disable structure around a MotorController or Enhanced-MotorController VI



In order to fix this, you must wire the device reference through the enabled state of the diagram disabled block

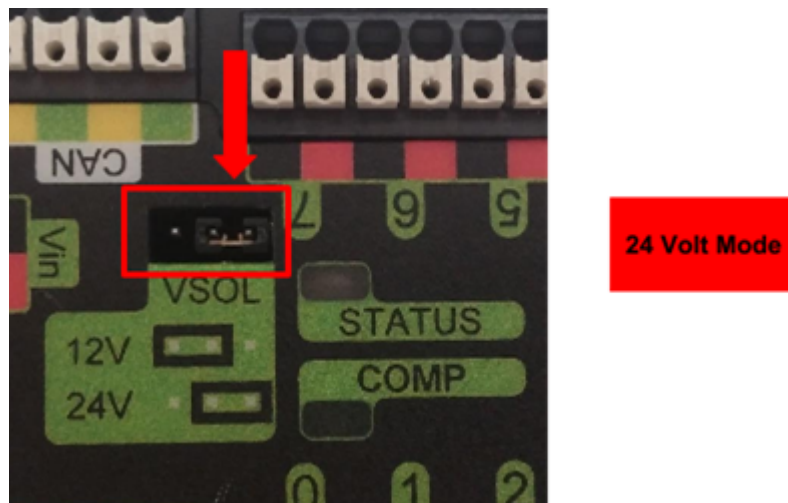




19.2 PCM

19.2.1 My compressor turns on and I have air pressure, but why isn't my solenoid turning on?

Check the red LED for the Solenoid channel. If the LED turns on as expected, make sure the Solenoid Voltage Jumper is set to the proper voltage (12 or 24 volts).



Warning: If you attempt to drive 12V Solenoids with 24V, you will damage the solenoids. If you attempt to drive 24V Solenoids with 12V, you *may* damage the solenoids.

19.2.2 Why isn't the Compressor turning on? Why does the PCM COMP LED not turn on?

In order for the compressor output to activate, certain conditions have to be met.

- The robot must be enabled.
- Robot software must have a pneumatics related object (compressor or solenoid).
- PCM must be powered/wired to CAN bus.
- PCM's device ID must match robot software.

If any of these conditions are not met, the compressor will not activate. The best method for root-causing wiring or software issues is to check the following conditions and symptoms in **sequential order**.

PCM must be powered.

This can be checked by ensuring the STATUS LED is illuminated. If the STATUS LED is off, recheck the power path from the PDP to the PCM. If using the fused output of the PDP, check the fuse. This can be done by removing the fuse and checking its continuity/DC-resistance, or simply by measuring the voltage across the power/ground wires that connect into the PCM's Vin Weidmuller input (should be approximately battery voltage or ~12V).

PCM must be on CAN Bus

The PCM must be connected to the CAN bus chain. If a PCM does not see a healthy CAN bus it will blink the STATUS LED red (See User's Guide for LED States).

Additionally the PCM will not appear in Phoenix Tuner or will report loss of communication. This is important to check because a red STATUS LED pattern may also reflect a fault condition (if robot is enabled). To distinguish a fault condition, confirm the PCM does appear in the configuration page, and use the Self-test Snapshot to identify which fault condition is occurring.

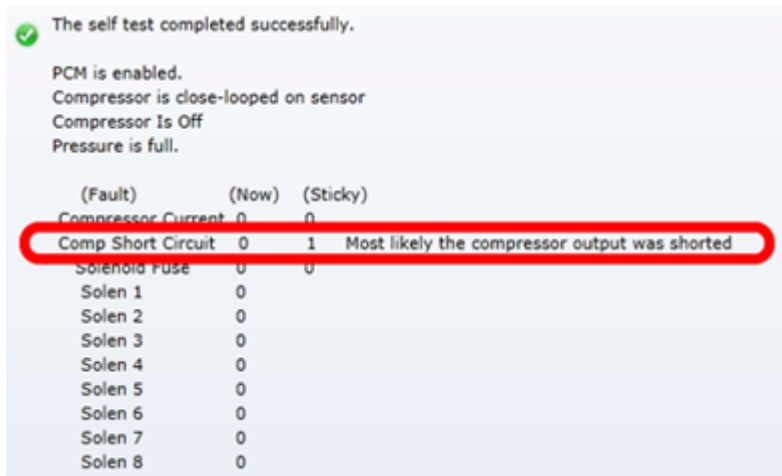
If these negative symptoms are noticed, recheck CAN bus harness and termination resistors. If several CAN devices are also blinking red then check the CANH/CANL chain. If it's just the PCM then inspect the Weidmuller CAN contacts on the PCM.

If the PCM CAN connection is healthy, then it should slowly blink green (when robot is disabled). It may blink orange instead to signal that a sticky fault has been logged. Use the Self-test Snapshot in Phoenix Tuner to inspect and clear sticky faults.

More information on faults and sticky faults is available under Faults-pcm.

Confirm PCM is not faulting.

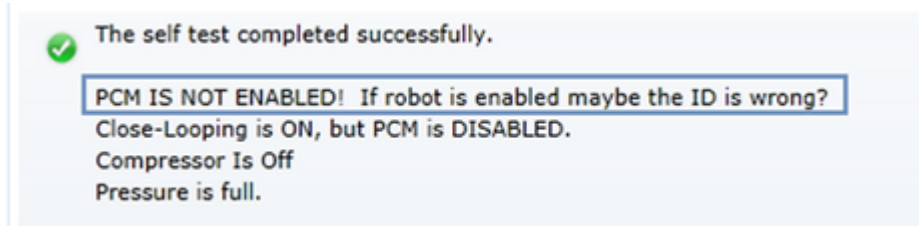
At this point the PCM should appear in the Phoenix Tuner CAN Devices tab. Using the Self-test Snapshot, determine if any faults are occurring "Now". Checking the sticky faults can also be helpful for identifying recent faults.



More information on faults and sticky faults is available under Faults-pcm.

The Robot must be enabled, Robot Software must create a pneumatics related object.

The PCM should appear in the Phoenix Tuner CAN Devices tab, however when enabling the robot, the STATUS LED may not transition to strobe green. Additionally, when performing the Self-test Snapshot, the report may read “PCM IS NOT ENABLED”



This is typical if the robot is not enabled OR if the robot application did not create any Solenoid or Compressor objects. This is how the programming API signals the intent of using pneumatics, and thus enabling the PCM.

Make sure the robot is truly enabled by looking at the Driver Station.

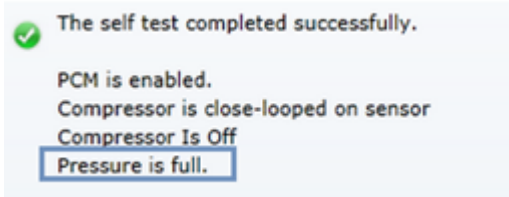
Instructions for creating a Solenoid, DoubleSolenoid or Compressor object in LabVIEW, C++, and Java can be found at <https://docs.wpilib.org>, (search for keyword “PCM”). Creating a single object of any pneumatics related type is sufficient for enabling the PCM (and therefore enabling compressor closed-loop).

Note: In order to create a software object for Solenoid or Compressor, typically the caller may specify the CAN Device ID (not specifying it typically defaults to selecting Device ID zero). This value must match what is specified in Phoenix Tuner. For more information see [Set Device IDs](#).

Tip: Since PCMs default with a device ID of zero, users only using one PCM may prefer to leave the default device ID. PCM Device ID range is allowed to overlap with the device ID of other non-PCM CAN devices.

Pressure Switch must be wired and must signal “not full”.

Even though a robot and PCM are enabled, the compressor output will not activate if the pressure switch is not connected or is indicating full pressure. The only way to inspect this reliably is to perform the Self-test Snapshot in Phoenix Tuner.



If Self-test Snapshot is reading “pressure is full” when the pressure gauge clearly is not full, recheck the wiring on the pressure switch and PCM.

The COMP LED must illuminate green.

If the COMP LED is off then the PCM is not activating the compressor output. The Self-test Snapshot is the best method for determining why. If the PCM is not present in the Phoenix Tuner recheck section the first 2 steps of this process. If the PCM is present and not enabled, recheck the robot program. If the Compressor is not “close-looped on sensor”, then the robot application must be using programming API to disable it. If pressure is erroneously reading “full”, recheck the previous step.

Compressor must be wired and functional.

If the COMP LED is illuminated green but the compressor still is not activating, then there may be a wiring issue between the PCM and the compressor. A voltmeter can be used to confirm that the PCM is applying 12V across the high and low side compressor output, and that 12V is reaching the compressor.

20.1 Imaging your Classmate (Veteran Image Download)

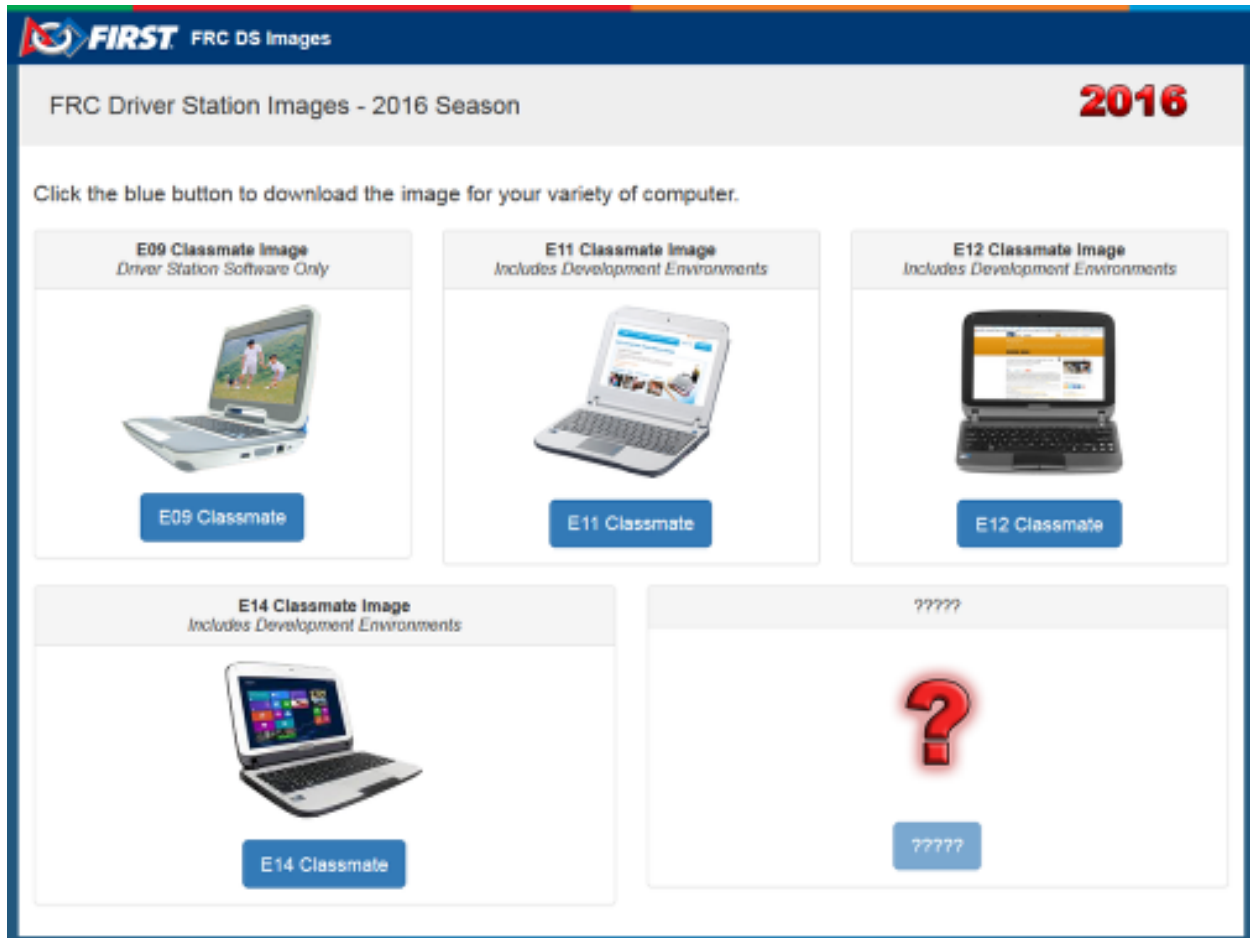
Note: Veteran teams are not required to re-image their classmate

This document describes the procedure for creating a bootable USB drive to restore the FRC image on a Classmate computer. If you do not wish to re-image your Classmate then you can start with the appropriate document for C++/Java, LabVIEW, or DS only.

20.1.1 Prerequisites

1. E09, E11, E12, or E14 Classmate computer or Acer ES1 computer
2. 16GB or larger USB drive
3. 7-Zip software installed (download [here](#)). As of the writing of this document, the current released version is 19.00 (2019-02-21).
4. RMprepUSB software installed (download [here](#)). Scroll down the page and select the stable (Full) version download link. As of the writing of this document, the current stable version is 2.1.743e.

20.1.2 Download the Computer Image



Download the image from the [FIRST FRC Driver Station System Image Portal](#). There are several computer images available, one for each model. On the download site, select the option that matches your computer by clicking the button below the image. Due to the limited size of the hard drive in the E09, it is supported with a DS/Utilities image only and does not have the IDEs for LabVIEW or C++/Java installed. All other images have the LabVIEW base installation already present.

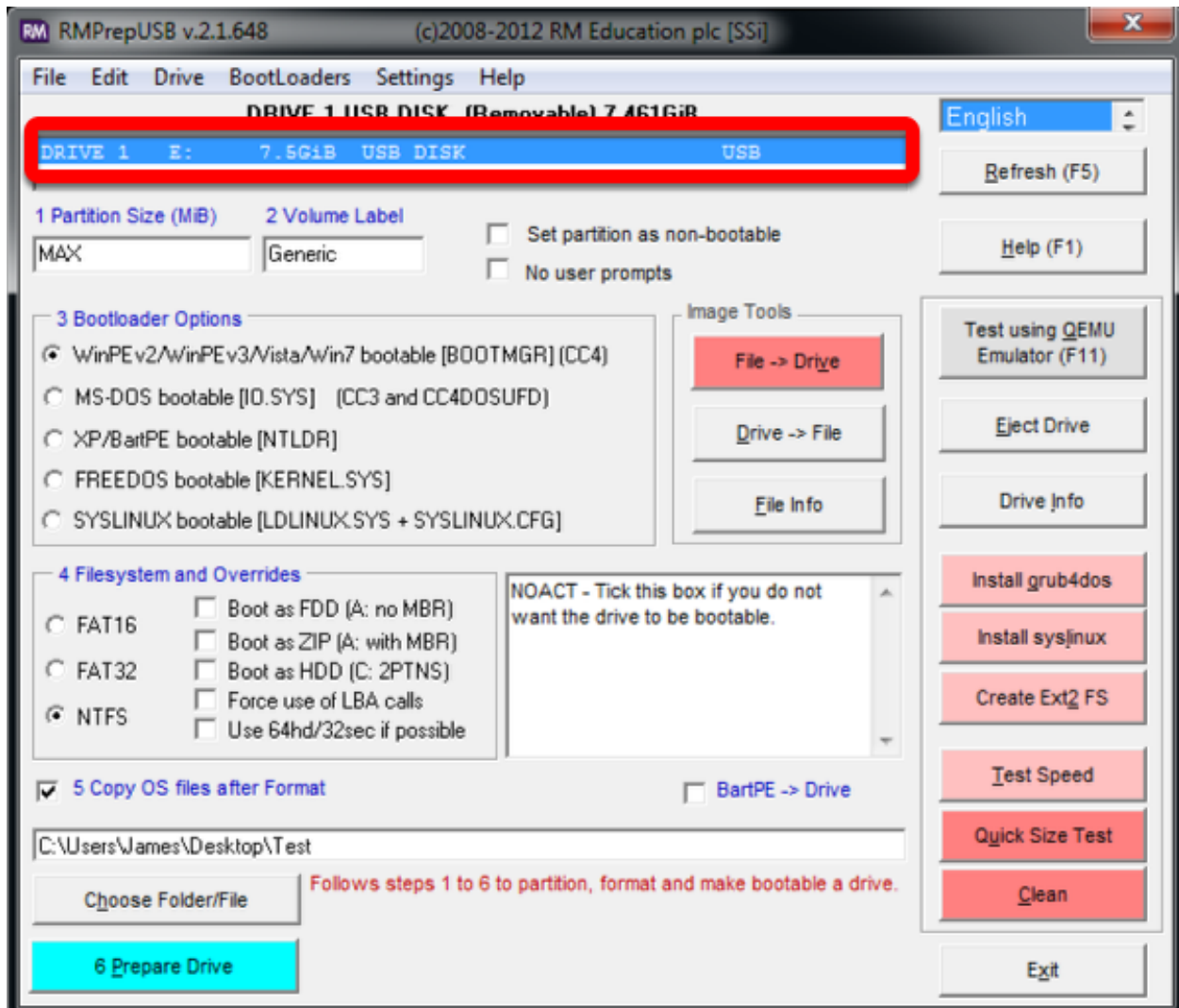
Note: These images only install the prerequisite core FRC software, it is still necessary to install the FRC specific updates. See the Update Software step for more information.

Warning: Due to computer availability, the E14 image provided is the 2018 image. If using this image, teams may need to remove the old IDE (LabVIEW or Eclipse) and install the new IDE.

20.1.3 Preparation

1. Place the image file downloaded from the site to a folder on your root drive (e.g. C:\2016_Image).
2. Connect 16GB or larger USB Flash drive to the PC to use as the new restoration drive.

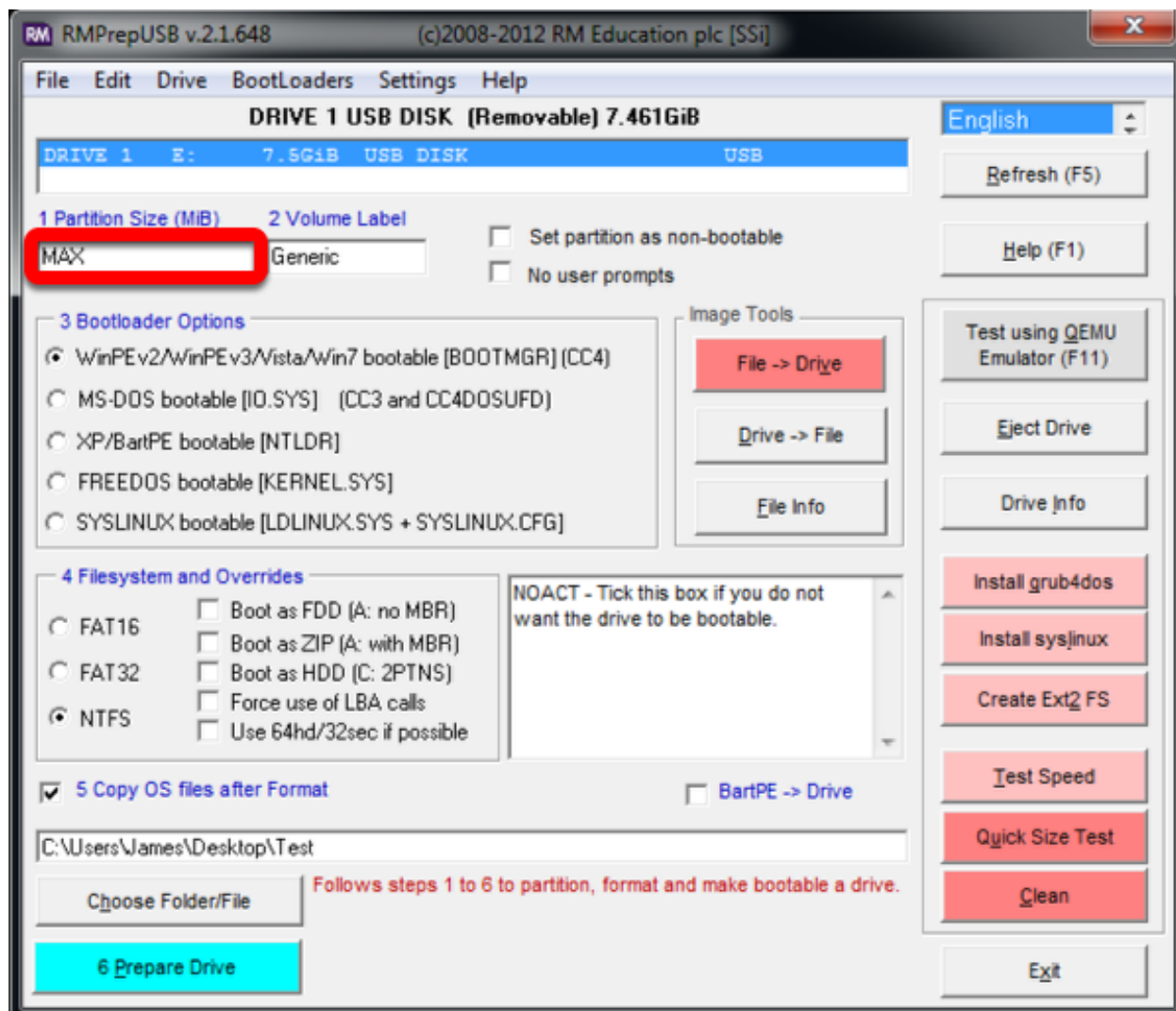
20.1.4 RMPrep



Start/Run RMPrepUSB

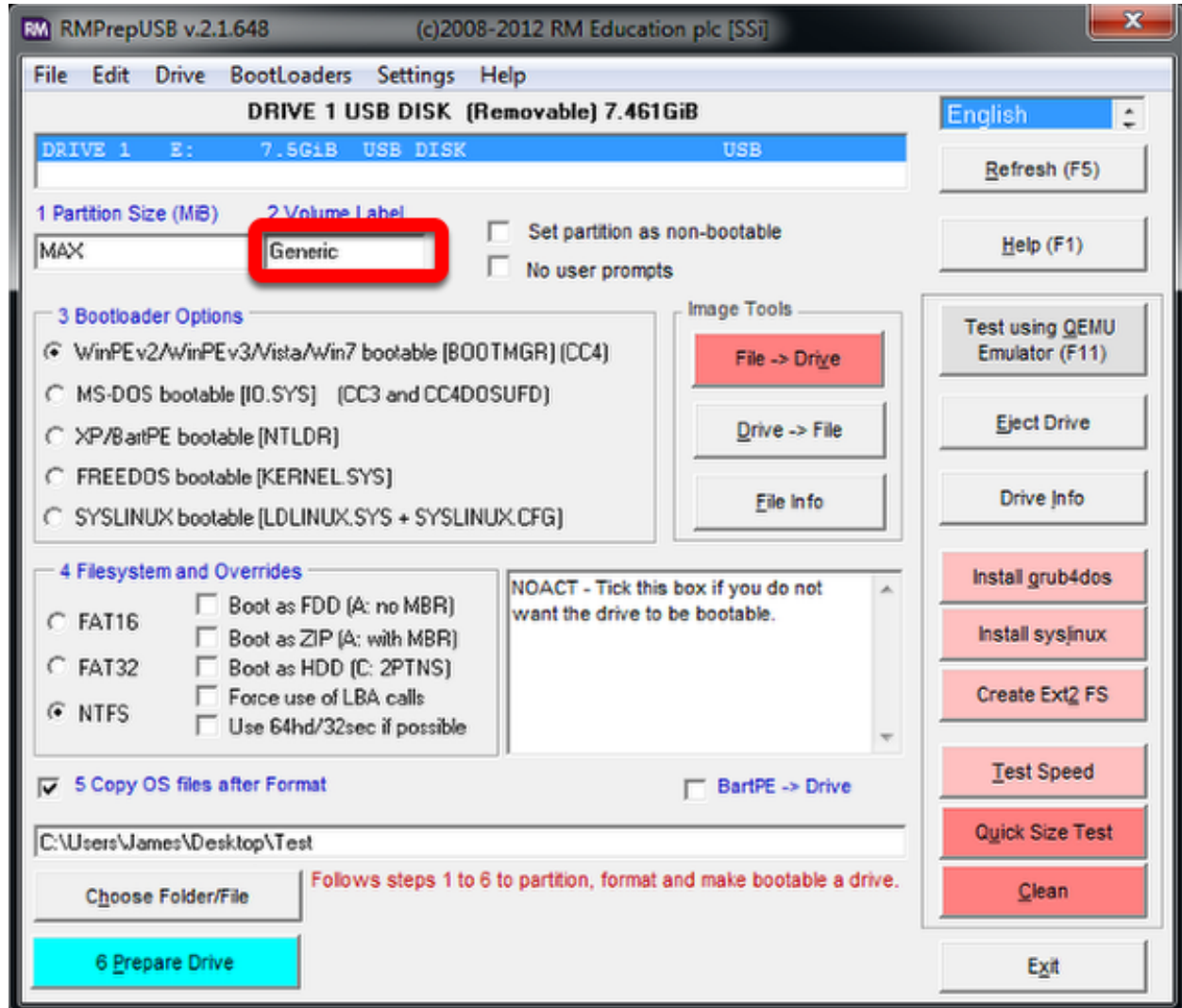
Select USB Drive

Set Partition Size



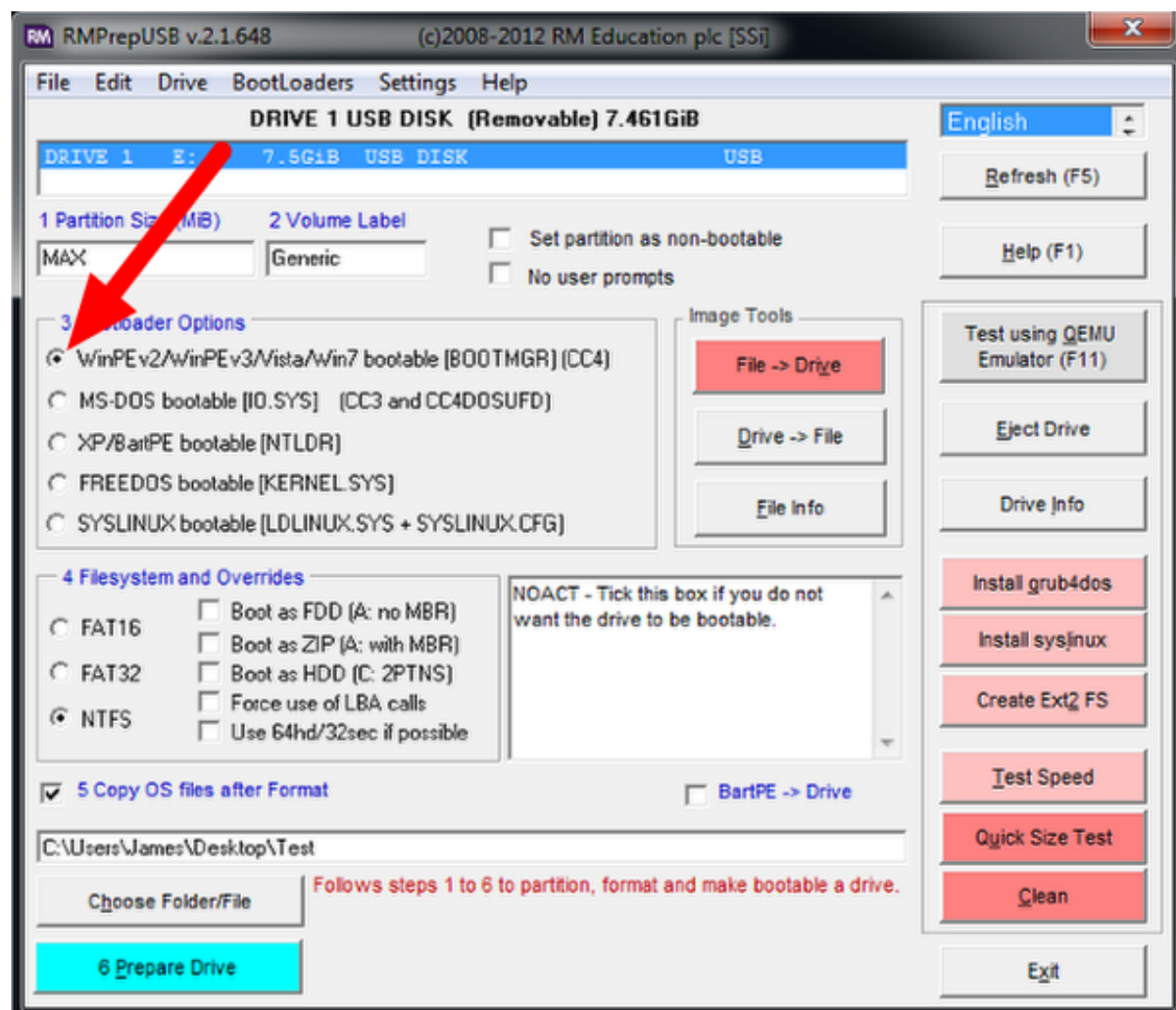
Set Partition Size to MAX

Set Volume Label



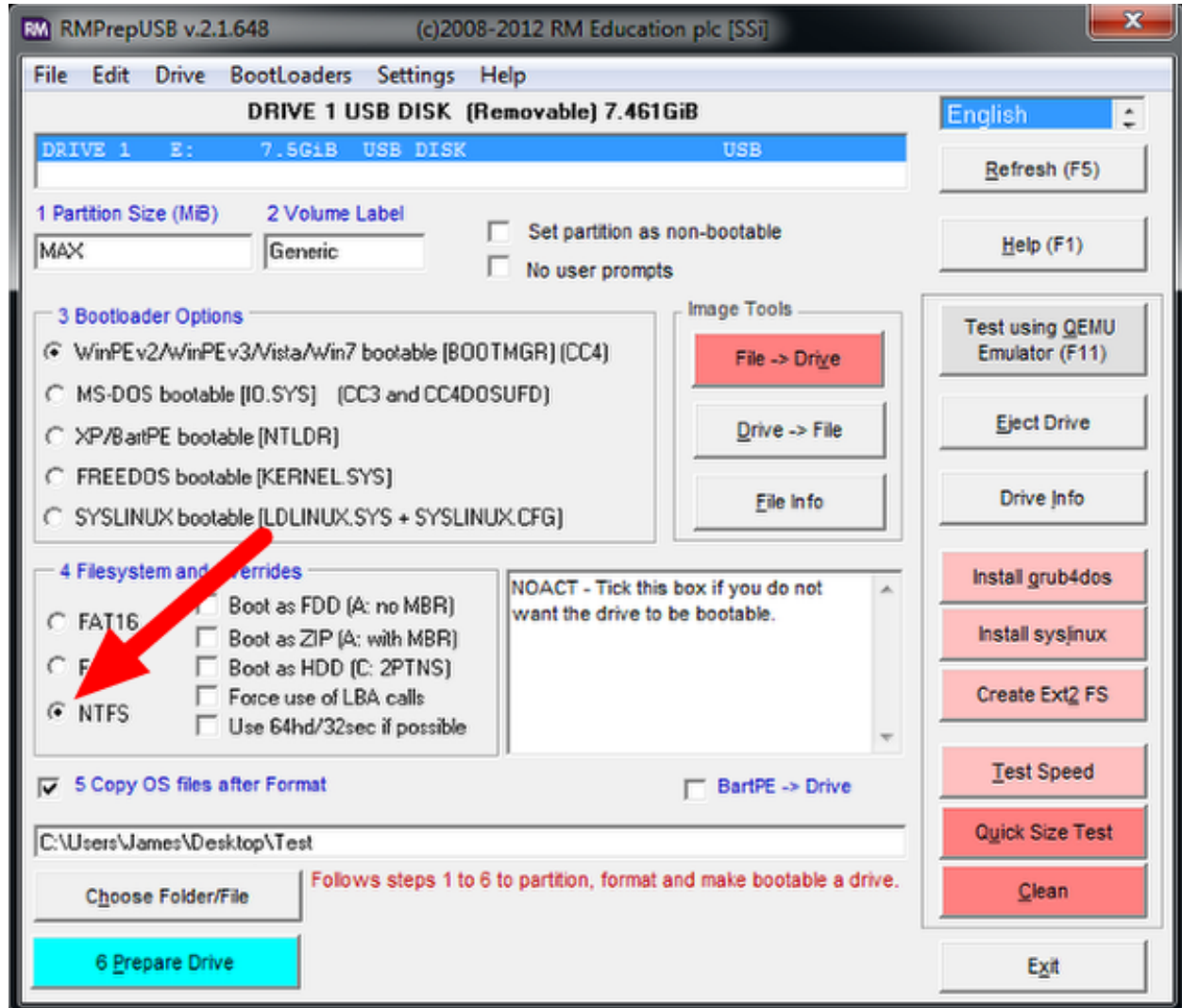
Set Volume Label to Generic

Set Bootloader Option



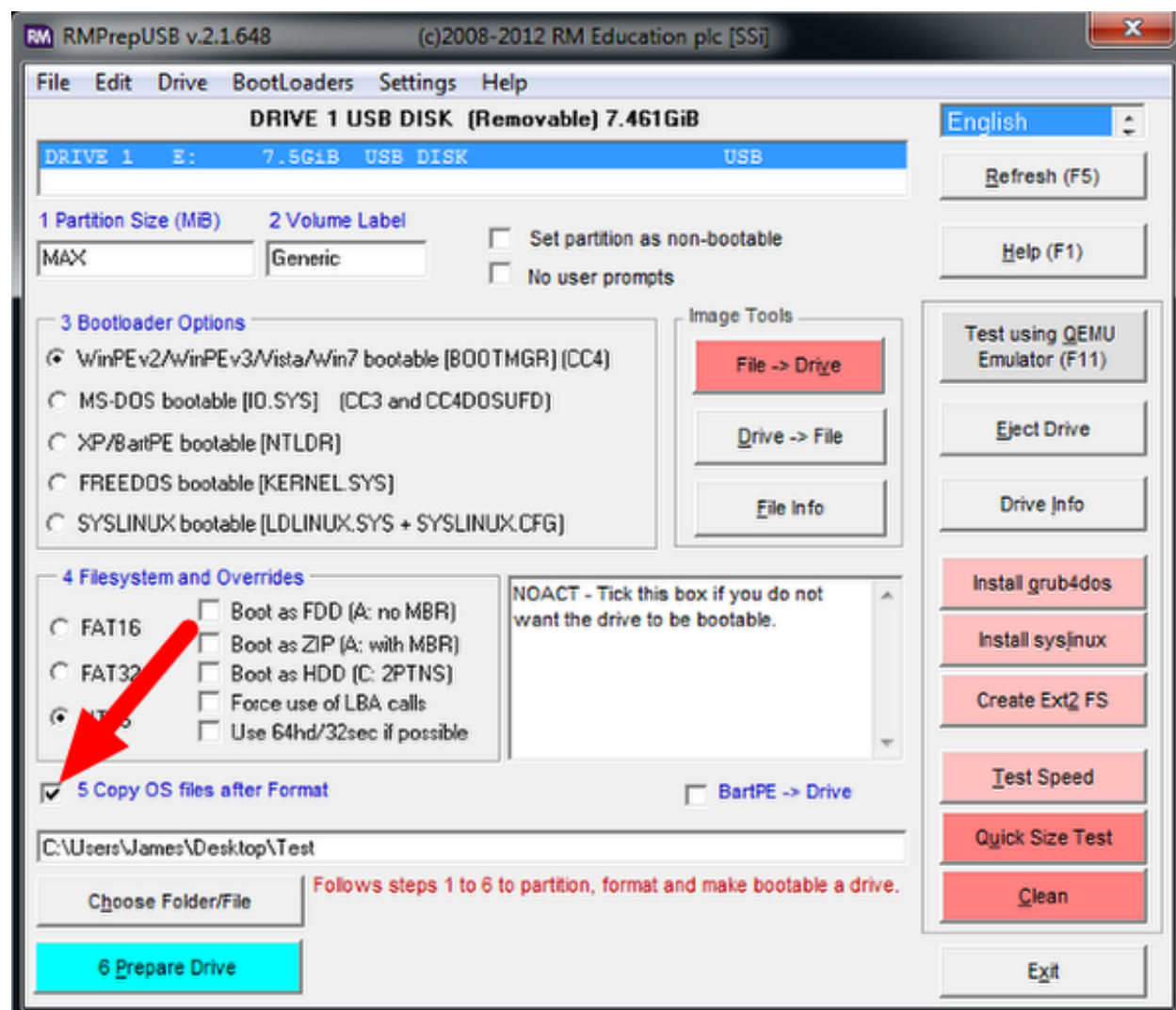
Select Bootloader Option “WinPE v2/WinPE v3/Vista/Win7 bootable”

Select Filesystem



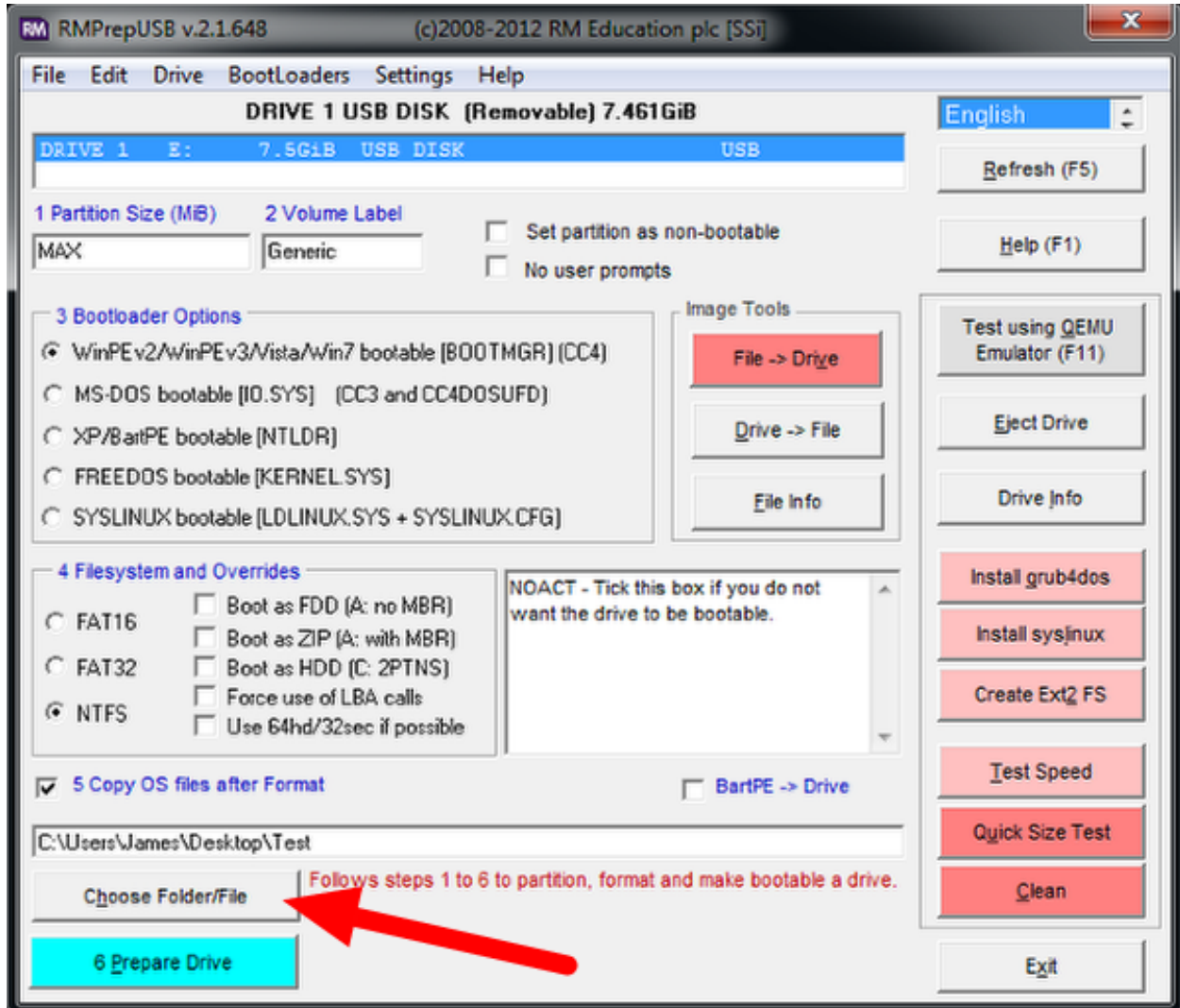
Select NTFS Filesystem

Copy OS Files Option



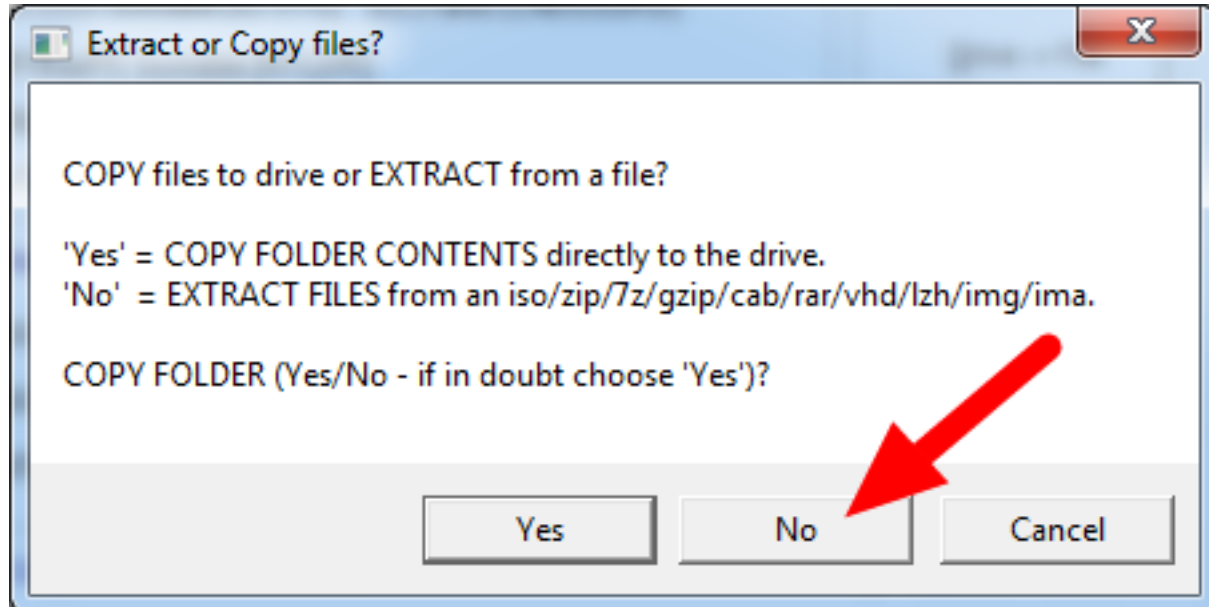
Ensure the “Copy OS files after Format” box is checked

Locate Image



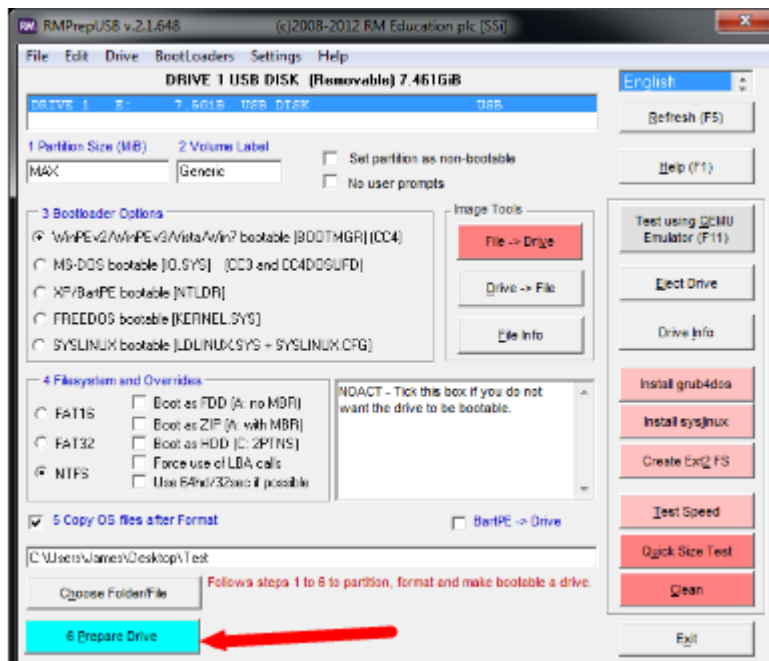
Select the "Choose Folder/File" button

Copy Files Dialog

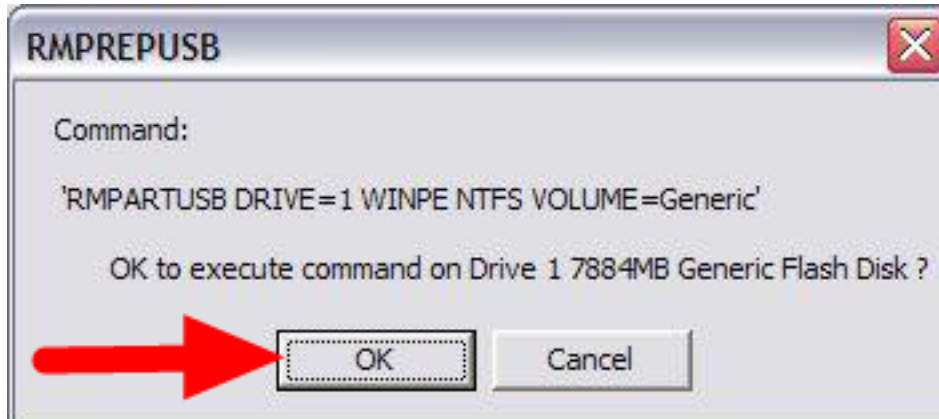


Choose “No” and select your .7z image

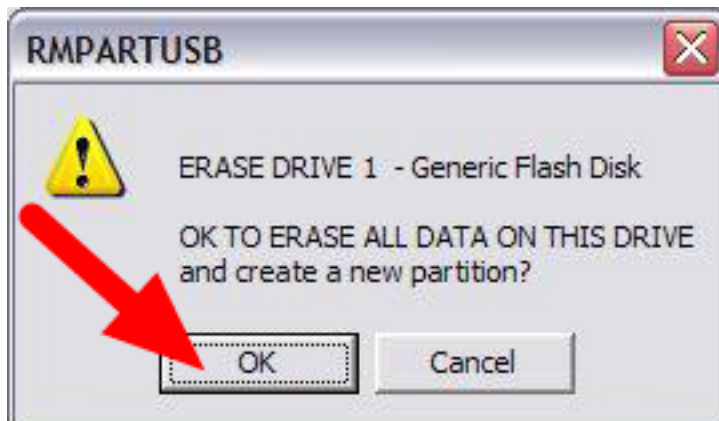
Prepare Drive



All configuration settings are now complete. Select “Prepare Drive” to begin the process

Confirmation Dialog 1

Click "OK" to execute the command on the selected USB Flash drive. A Command Prompt will open showing the progress

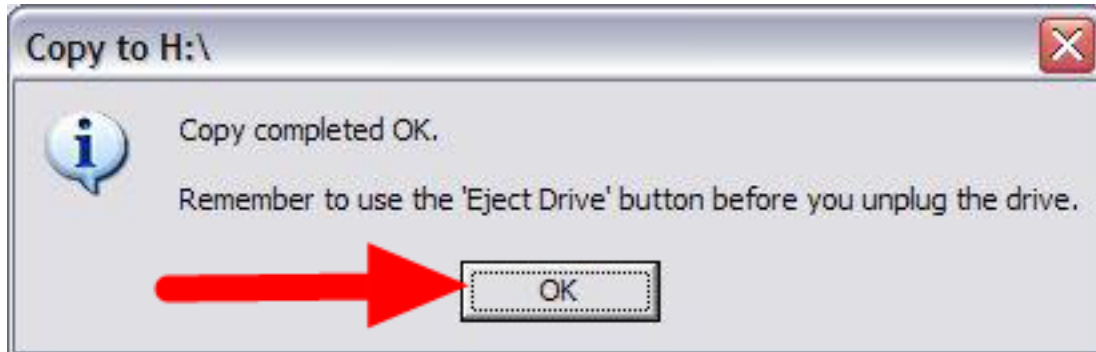
Confirmation Dialog 2

Click "OK" to format the USB drive

Danger: ALL DATA ON THE DRIVE WILL BE ERASED!

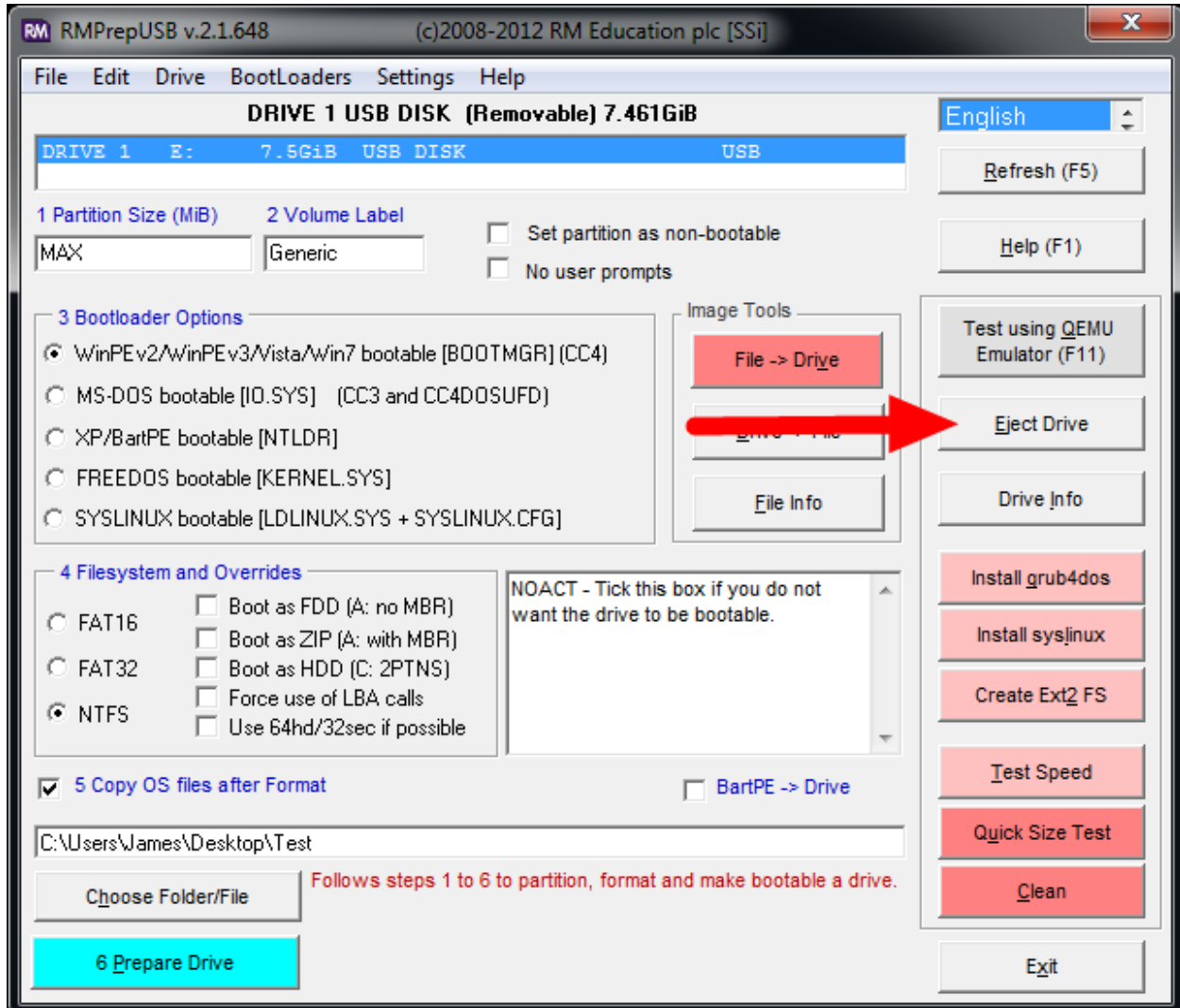
Decryption

Note: If you are using an encrypted version of the image downloaded before kickoff you will be prompted to enter the decryption key found at the end of the Kickoff video.

Copy Complete

Once formatting is complete, the restoration files will be extracted and copied to the USB drive. This process should take ~15 minutes when connected to a USB 2.0 port. When all files have been copied, this message will appear, press OK to continue.

Eject Drive

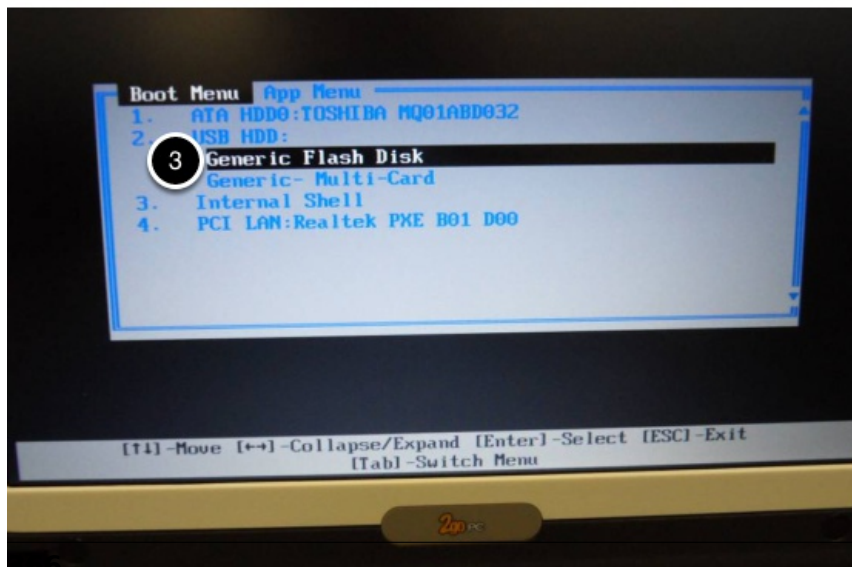
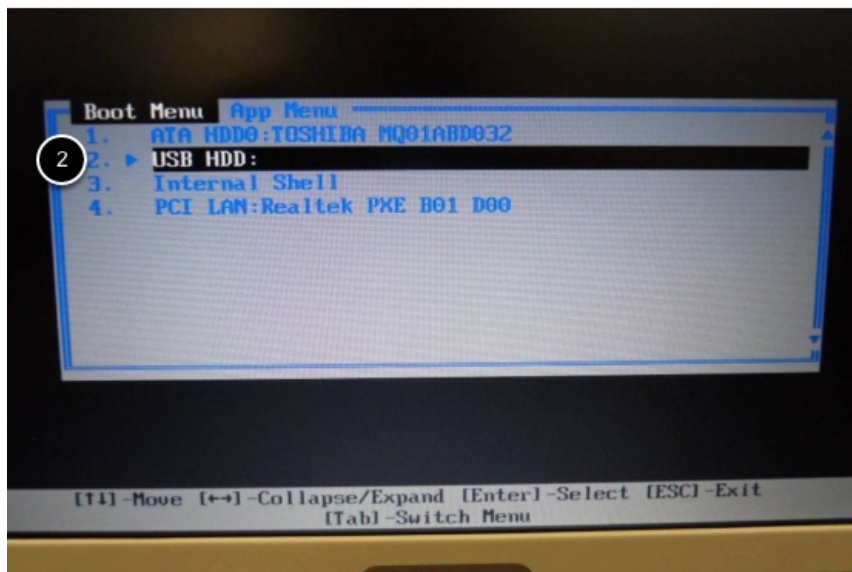
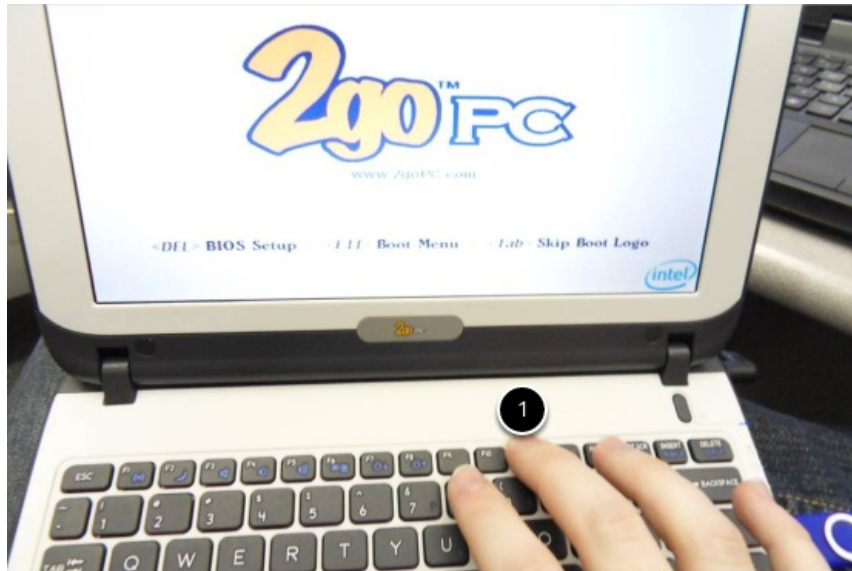


Press the “Eject Drive” button to safely remove the USB drive. The USB drive is now ready to be used to restore the image onto the PC.

20.1.5 Hardware Setup

1. Make sure the computer is turned off, but plugged in.
2. Insert the USB Thumb Drive into a USB port on the Driver Station computer.

Boot to USB



Classmate:

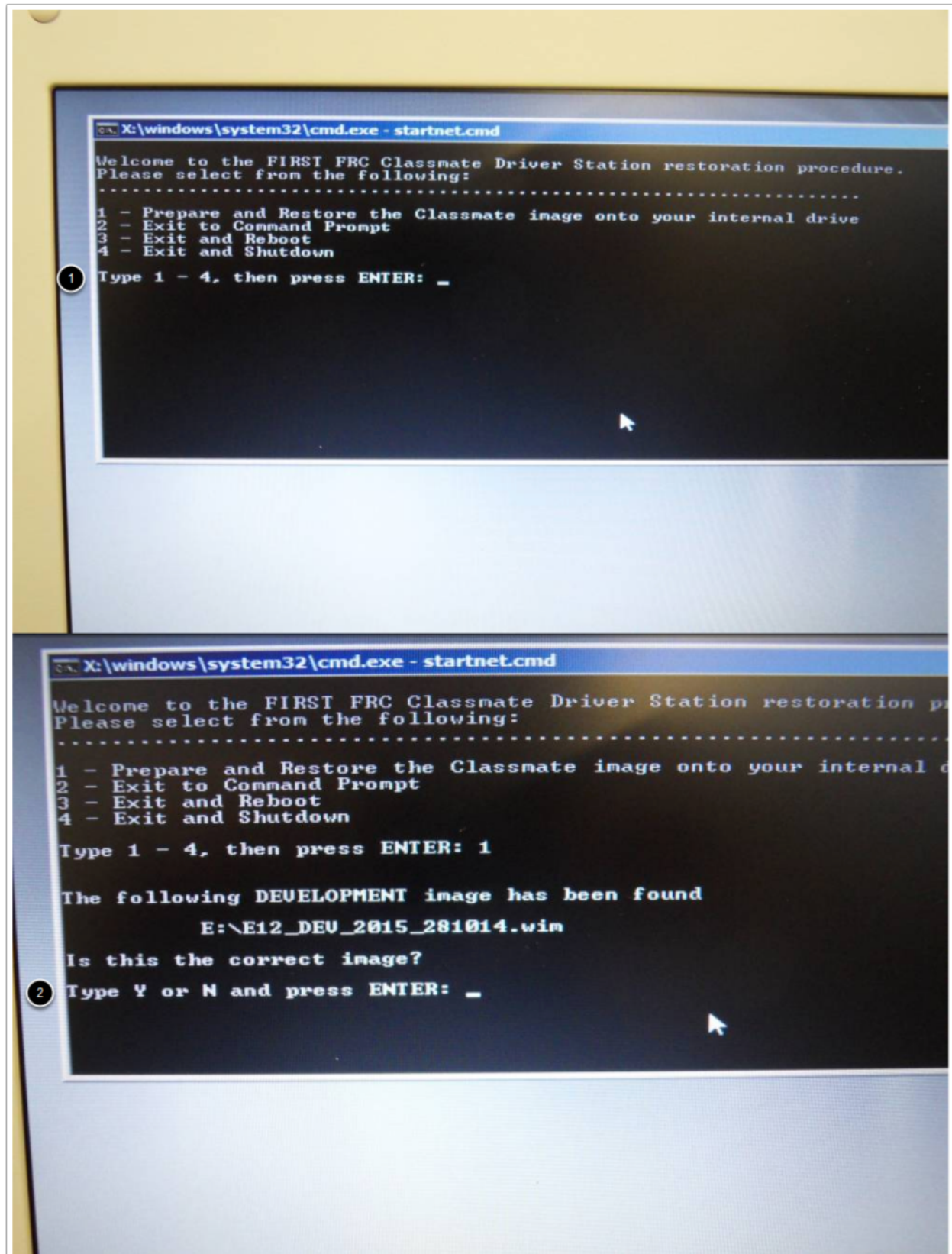
1. Power on the Classmate and tap the F11 key on the keyboard. Tapping the F11 key during boot will bring up the boot menu.
2. Use the up/down keys to select the **USB HDD:** entry on the menu, then press the right arrow to expand the listing
3. Use the up/down arrow keys on the keyboard to select the USB device (it will be called "Generic Flash Disk"). Press the ENTER key when the USB device is highlighted.

Acer ES1:

1. Power on the computer and tap the F12 key on the keyboard. Tapping the F12 key during boot will bring up the boot menu.
2. Use the up/down keys to select the **USB HDD: Generic** entry on the menu, then press the ENTER key when the USB device is highlighted.

Acer ES1: If pressing F12 does not pull up the boot menu or if the USB device is not listed in the boot menu, see "Checking BIOS Settings" at the bottom of this article.

Image the Classmate



1. To confirm that you want to reimage the Classmate, type “1” and press ENTER.
2. Then, type “Y” and press ENTER. The Classmate will begin re-imaging. The installation will take 15-30 minutes.
3. When the installation is complete, remove the USB drive.
4. Restart the Classmate. The Classmate will boot into Windows.

20.1.6 Initial Driver Station Boot

The first time the Classmate is turned on, there are some unique steps, listed below, that you’ll need to take. The initial boot may take several minutes; make sure you do not cycle power during the process.

Note: These steps are only required during original startup.

Enter Setup

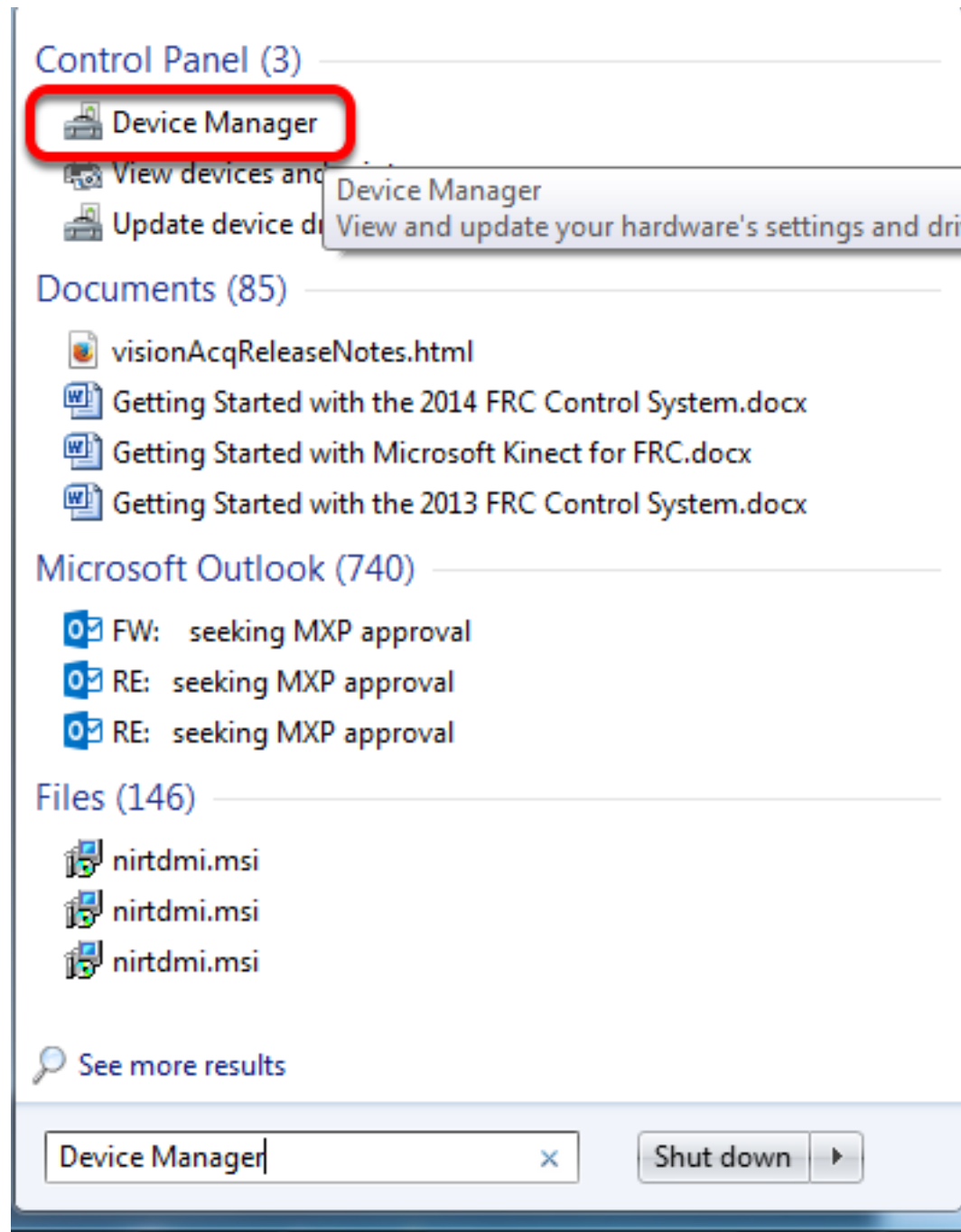
1. Log into the Developer account.
2. Click “Ask me later”.
3. Click “OK”. The computer now enters a Set Up that may take a few minutes.

Activate Windows

1. Establish an Internet connection.
2. Once you have an Internet connection, click the Start menu, right click “Computer” and click “Properties”.
3. Scroll to the bottom section, “Windows activation”, and Click “Activate Windows now”
4. Click “Activate Windows online now”. The activation may take a few minutes.
5. When the activation is complete, close all of the windows.

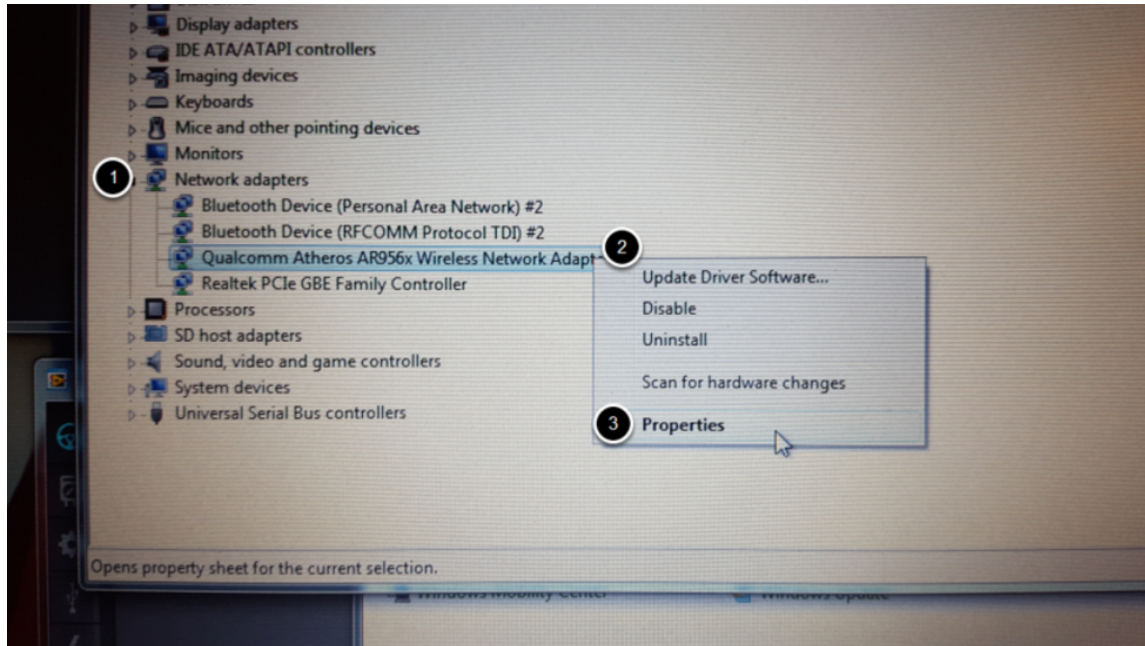
Microsoft Security Essentials

Navigate through the Microsoft Security Essentials Setup Wizard. Once it is complete, close all of the windows.

Acer ES1: Fix Wireless Driver**Acer ES1 PC only!**

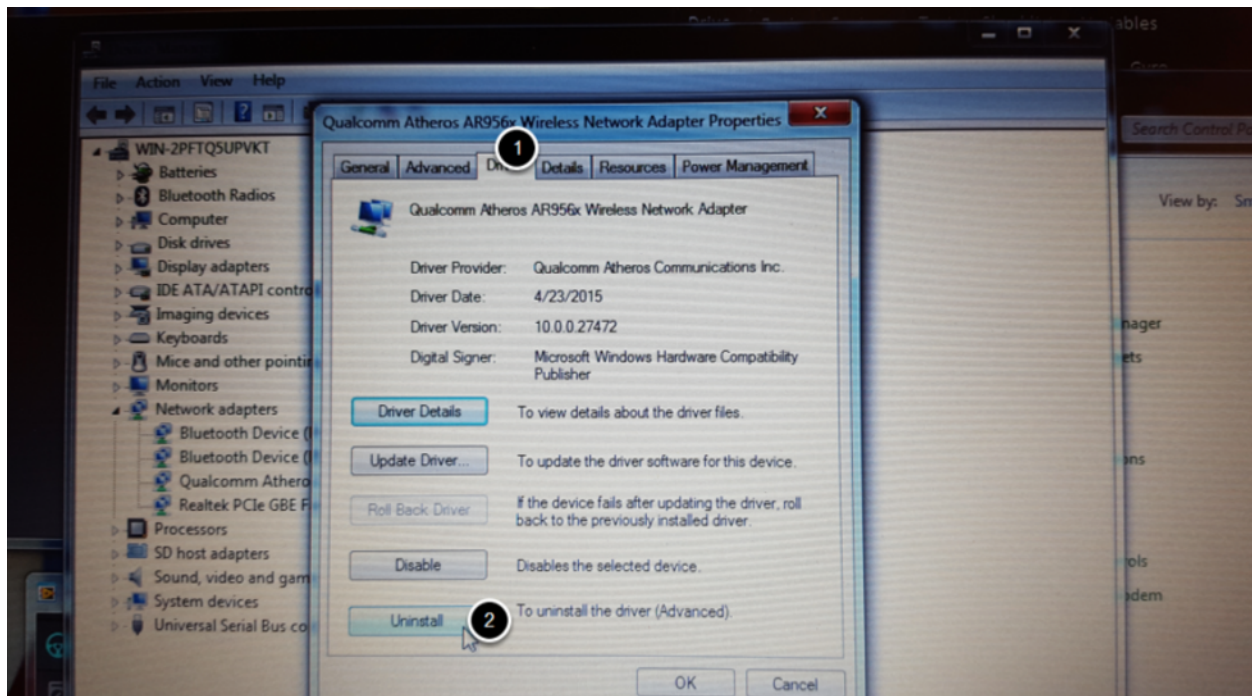
The default wireless driver in the image may have issues with intermittent communication with the robot radio. The correct driver is in the image, but could not be set to load by default. To load the correct driver, open the Device Manager by clicking start, typing "Device Manager" in the box and clicking Device Manager.

Open Wireless Device Properties



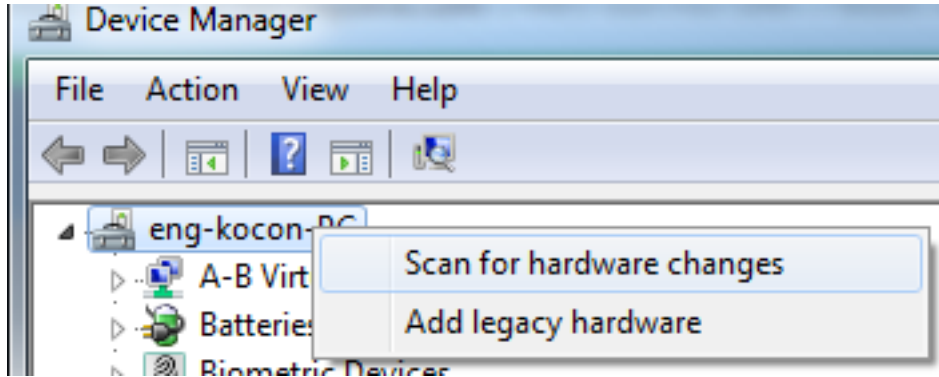
Click on the arrow next to Network Adapters to expand it and locate the Wireless Network Adapter. Right click the adapter and select Properties.

Uninstall-Driver



Click on the Driver tab, then click the Uninstall button. Click Yes at any prompts.

Scan for New Hardware

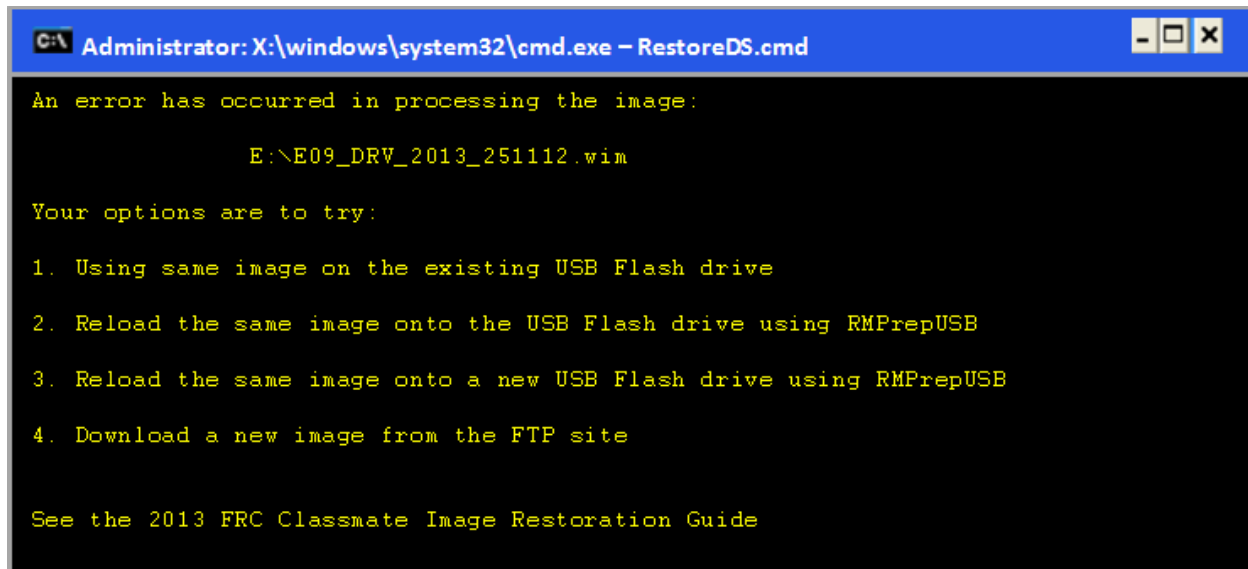


Right click on the top entry of the tree and click “Scan for hardware changes”. The wireless adapter should automatically be re-detected and the correct driver should be installed.

20.1.7 Update Software

In order for the Classmate images to be prepared on time, they are created before the final versions of the software were ready. To use the software for FRC some additional components will need to be installed. LabVIEW teams should continue with Installing the FRC Game Tools (All Languages). C++ or Java teams should continue Installing C++ and Java Development Tools for FRC.

20.1.8 Errors during Imaging Process



If an error is detected during the imaging process, the following screen will appear. Note that the screenshot below shows the error screen for the Driver Station-only image for the E09. The specific image filename shown will vary depending on the image being applied.

The typical reason for the appearance of this message is due to an error with the USB device on which the image is stored. Each option is listed below with further details as to the actions

you can take in pursuing a solution. Pressing any key once this error message is shown will return the user to the menu screen shown in Image the Classmate.

Option 1

Using same image on the existing USB Flash drive To try this option, press any key to return to the main menu and select #1. This will run the imaging process again.

Option 2

Reload the same image onto the USB Flash drive using RMPrepUSB It's possible the error message was displayed due to an error caused during the creation of the USB Flash drive (e.g. file copy error, data corruption, etc.) Press any key to return to the main menu and select #4 to safely shutdown the Classmate then follow the steps starting with RMPrep to create a new USB Restoration Key using the same USB Flash drive.

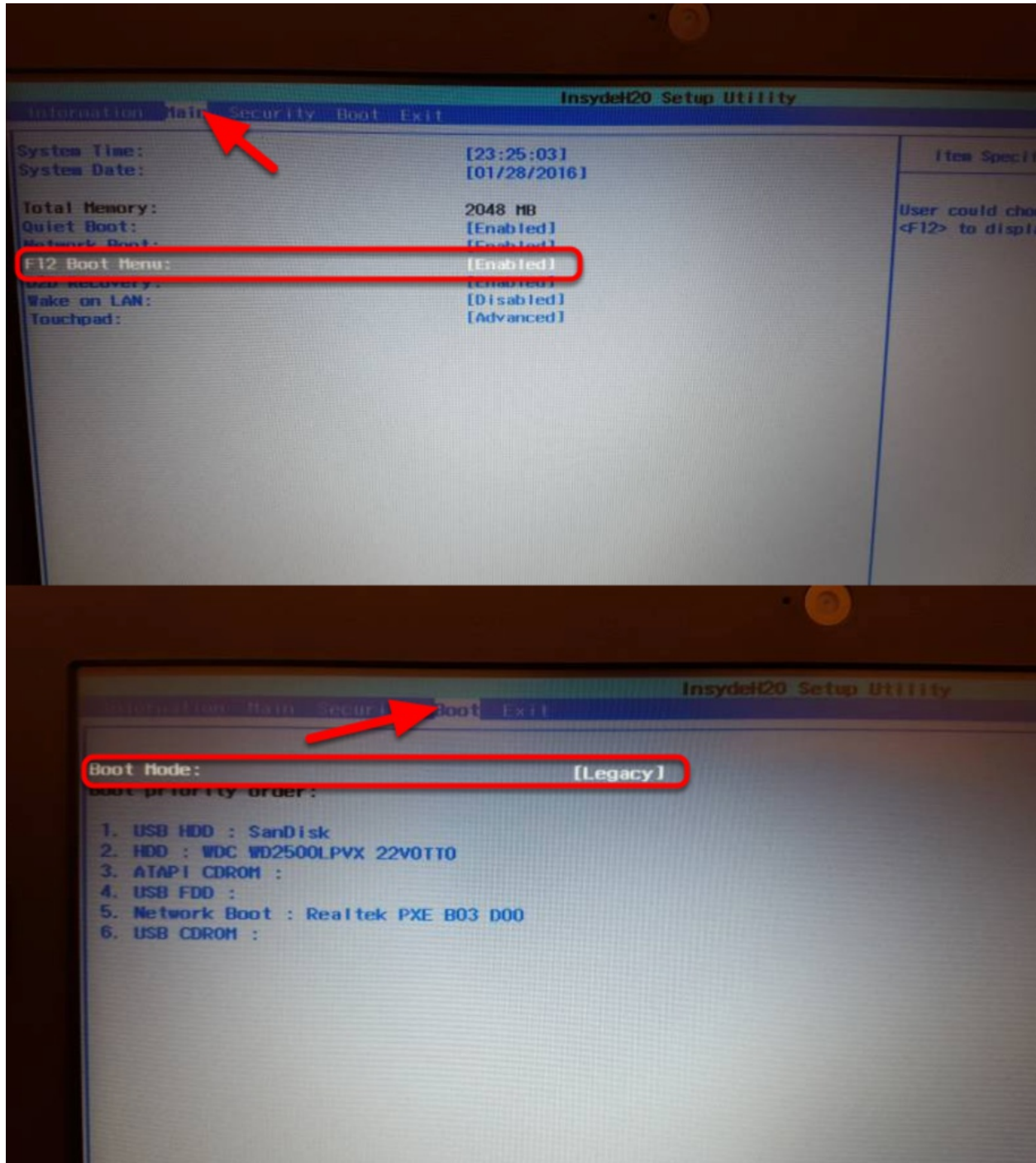
Option 3

Reload the same image onto a new USB Flash drive using RMPrepUSB The error message displayed may also be caused by an error with the USB Flash drive itself. Press any key to return to the main menu and select #4 to safely shutdown the Classmate. Select a new USB Flash drive and follow the steps starting with RMPrep.

Option 4

Download a new image An issue with the downloaded image may also cause an error when imaging. Press any key to return to the main menu and select #4 to safely shutdown the Classmate. Starting with Download the Classmate Image create a new copy of the imaging stick.

Checking BIOS Settings



If you are having difficulty booting to USB, check the BIOS settings to insure they are correct. To do this:

- Repeatedly tap the **F2** key while the computer is booting to enter the BIOS settings
- Once the BIOS settings screen has loaded, use the right and left arrow keys to select the "Main" tab, then check if the line for "F12 Boot Menu" is set to "Enabled". If it is not, use the Up/Down keys to highlight it, press Enter, use Up/Down to select "Enabled" and

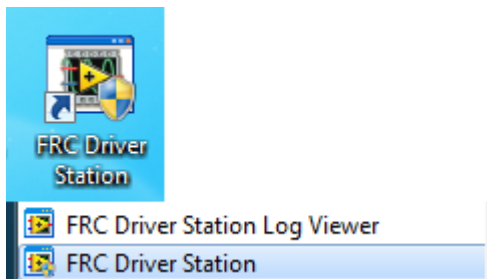
press Enter again.

- Next, use the Left/Right keys to select the “Boot” tab. Make sure that the “Boot Mode” is set to “Legacy”. If it is not, highlight it using UpDown, press Enter, highlight “Legacy” and press Enter again. Press Enter to move through any pop-up dialogs you may see.
- Press F10 to save any changes and exit.

20.2 FRC Driver Station Powered by NI LabVIEW

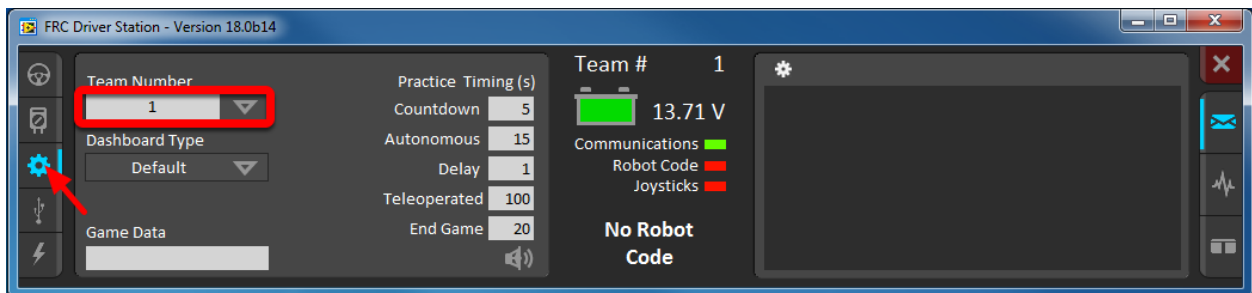
This article describes the use and features of the FRC Driver Station Powered by NI LabVIEW. For information on installing the Driver Station software see [this document](#).

20.2.1 Starting the FRC Driver Station



The FRC Driver Station can be launched by double-clicking the icon on the Desktop or by selecting Start->All Programs->FRC Driver Station.

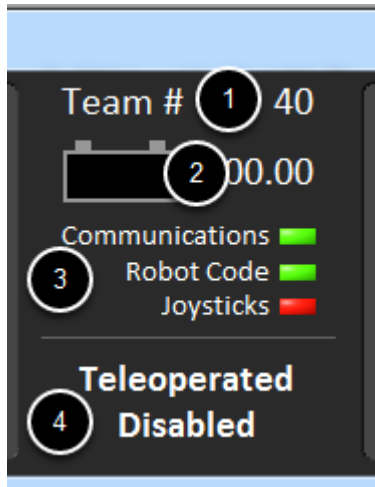
20.2.2 Setting Up the Driver Station



The DS must be set to your team number in order to connect to your robot. In order to do this click the Setup tab then enter your team number in the team number box. Press return or click outside the box for the setting to take effect.

PCs will typically have the correct network settings for the DS to connect to the robot already, but if not, make sure your Network adapter is set to DHCP.

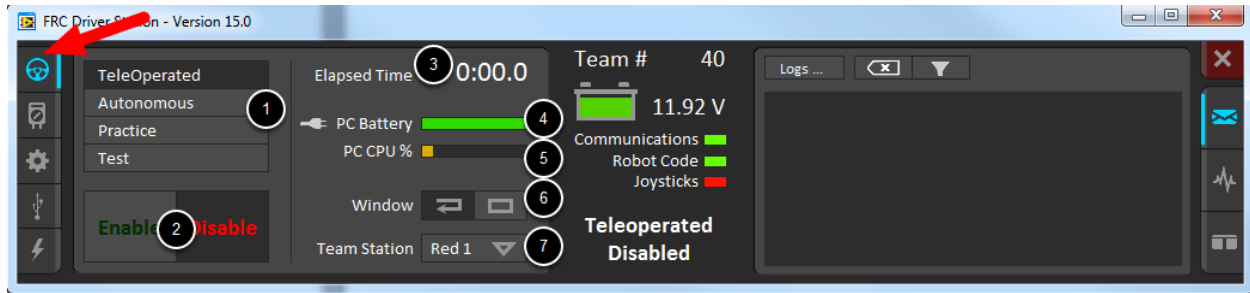
20.2.3 Status Pane



The Status Pane of the Driver Station is located in the center of the display and is always visible regardless of the tab selected. It displays a selection of critical information about the state of the DS and robot:

1. **Team #** - The Team number the DS is currently configured for. This should match your FRC team number. To change the team number see the Setup Tab.
2. **Battery Voltage** - If the DS is connected and communicating with the roboRIO this displays current battery voltage as a number and with a small chart of voltage over time in the battery icon. The background of the numeric indicator will turn red when the roboRIO brownout is triggered. See </docs/software/roborio-info/roborio-brownouts> for more information.
3. **Major Status Indicators** - These three indicators display major status items for the DS. The “Communications” indicates whether the DS is currently communicating with the FRC Network Communications Task on the roboRIO (it is split in half for the TCP and UDP communication). The “Robot Code” indicator shows whether the team Robot Code is currently running (determined by whether or not the Driver Station Task in the robot code is updating the battery voltage), The “Joysticks” indicator shows if at least one joystick is plugged in and recognized by the DS.
4. **Status String** - The Status String provides an overall status message indicating the state of the robot, some examples are “No Robot Communication”, “No Robot Code”, “Emergency Stopped”, and “Teleoperated Enabled”. When the roboRIO brownout is triggered this will display “Voltage Brownout”.

20.2.4 Operation Tab

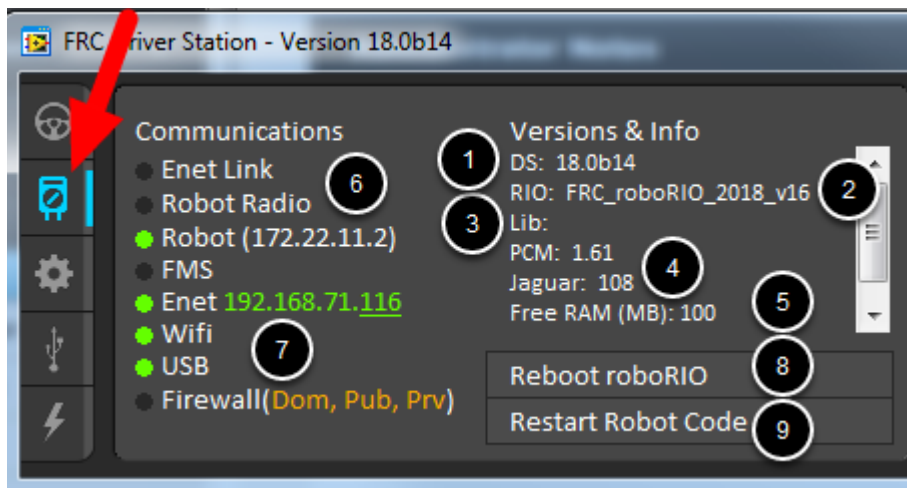


The Operations Tab is used to control the mode of the robot and provide additional key status indicators while the robot is running.

1. Robot Mode - This section controls the Robot Mode. Practice Mode causes the robot to cycle through the same transitions as an FRC match after the Enable button is pressed (timing for practice mode can be found on the setup tab).
2. Enable/Disable - These controls enable and disable the robot. See also [Driver Station Key Shortcuts](#)
3. Elapsed Time - Indicates the amount of time the robot has been enabled
4. PC Battery - Indicates current state of DS PC battery and whether the PC is plugged in
5. PC CPU% - Indicates the CPU Utilization of the DS PC
6. Window Mode - When not on the Driver account on the Classmate allows the user to toggle between floating (arrow) and docked (rectangle)
7. Team Station - When not connected to FMS, sets the team station to transmit to the robot.

Note: When connected to the Field Management System the controls in sections 1, and 2 will be replaced by the words FMS Connected and the control in Section 7 will be greyed out.

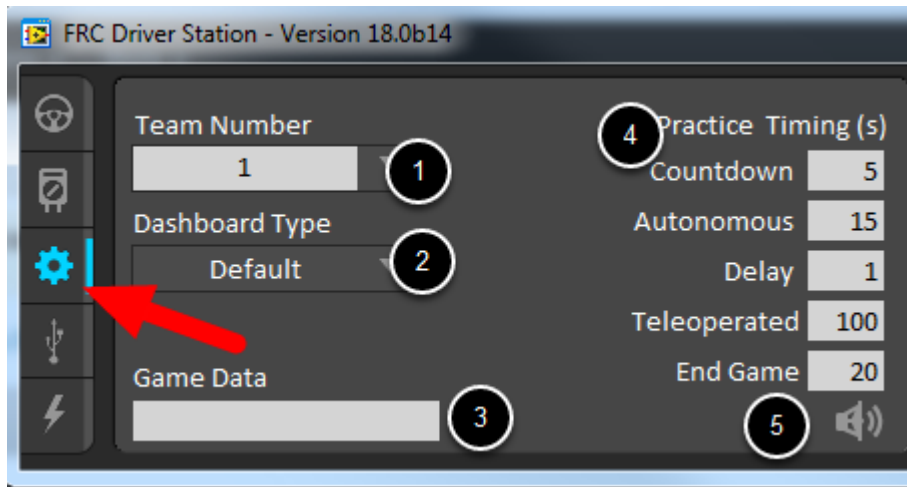
20.2.5 Diagnostics Tab



The Diagnostics Tab contains additional status indicators that teams can use to diagnose issues with their robot:

1. DS Version - Indicates the Driver Station Version number
2. roboRIO Image Version - String indicating the version of the roboRIO Image
3. WPILib Version - String indicating the version of WPILib in use
4. CAN Device Versions - String indicating the firmware version of devices connected to the CAN bus. These items may not be present if the CTRE Phoenix framework has not been loaded
5. Memory Stats - This section shows stats about the roboRIO memory
6. Connection Indicators - The top half of these indicators show connection status to various components.
 - “Enet Link” indicates the computer has something connected to the ethernet port.
 - “Robot Radio” indicates the ping status to the robot wireless bridge at 10.XX.YY.1.
 - “Robot” indicates the ping status to the roboRIO using mDNS (with a fallback of a static 10.TE.AM.2 address).
 - “FMS” indicates if the DS is receiving packets from FMS (this is NOT a ping indicator).
7. Network Indicators - The second section of indicators indicates status of network adapters and firewalls. These are provided for informational purposes, communication may be established with one or more unlit indicators in this section
 - “Enet” indicates the IP address of the detected Ethernet adapter
 - “WiFi” indicates if a wireless adapter has been detected as enabled
 - “USB” indicates if a roboRIO USB connection has been detected
 - “Firewall” indicates if any firewalls are detected as enabled. Enabled firewalls will show in orange (Dom = Domain, Pub = Public, Prv = Private)
8. Reboot roboRIO - This button attempts to perform a remote reboot of the roboRIO (after clicking through a confirmation dialog)
9. Restart Robot Code - This button attempts to restart the code running on the robot (but not restart the OS)

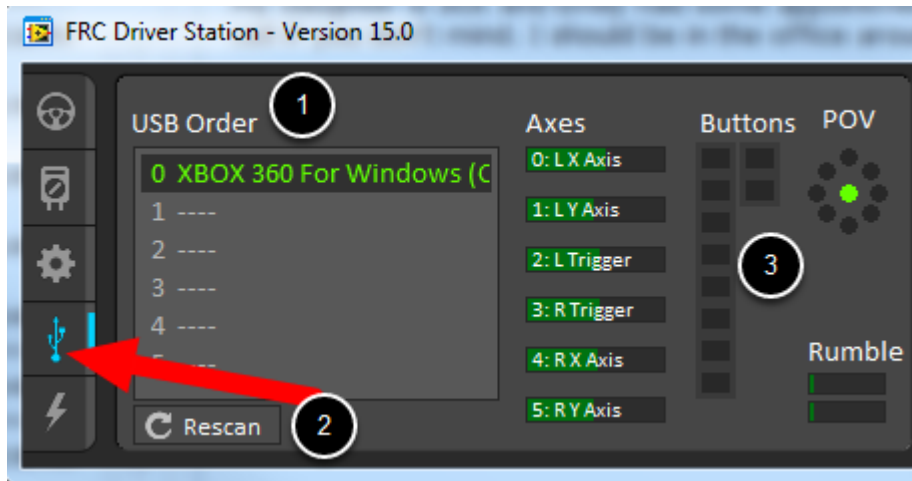
20.2.6 Setup Tab



The Setup Tab contains a number of buttons teams can use to control the operation of the Driver Station:

1. Team Number - Should contain your FRC Team Number. This controls the mDNS name that the DS expects the robot to be at. Shift clicking on the dropdown arrow will show all roboRIO names detected on the network for troubleshooting purposes.
2. Dashboard Type - Controls what Dashboard is launched by the Driver Station. Default launches the file pointed to by the "FRC DS Data Storage.ini" file, by default this is Dashboard.exe in the Program Files\FRC Dashboard folder. LabVIEW attempts to launch a dashboard at the default location for a custom built LabVIEW dashboard, but will fall back to the default if no dashboard is found. SmartDashboard and Shuffleboard launch the respective dashboards included with the C++ and Java WPILib installation.
3. Game Data - This box can be used for at home testing of the Game Data API. Text entered into this box will appear in the Game Data API on the Robot Side. When connected to FMS, this data will be populated by the field automatically.
4. Practice Mode Timing - These boxes control the timing of each portion of the practice mode sequence. When the robot is enabled in practice mode the DS automatically proceeds through the modes indicated from top to bottom.
5. Audio Control - This button controls whether audio tones are sounded when the Practice Mode is used.

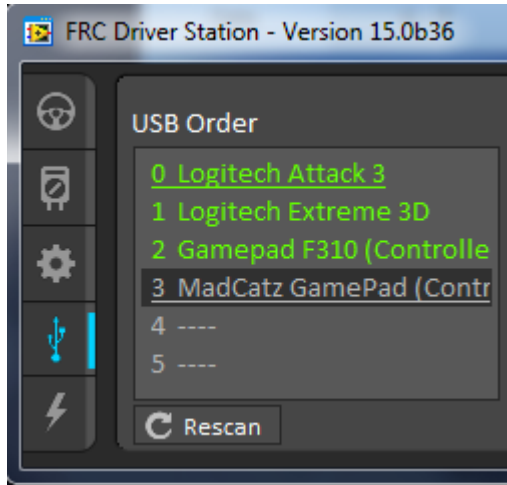
20.2.7 USB Devices Tab



The USB Devices tab includes the information about the USB Devices connected to the DS

1. USB Setup List - This contains a list of all compatible USB devices connected to the DS. Pressing a button on a device will highlight the name in green and put 2 *s before the device name
2. Rescan - This button will force a Rescan of the USB devices. While the robot is disabled, the DS will automatically scan for new devices and add them to the list. To force a complete re-scan or to re-scan while the robot is Enabled (such as when connected to FMS during a match) press F1 or use this button.
3. Device indicators - These indicators show the current status of the Axes, buttons and POV of the joystick.
4. Rumble - For XInput devices (such as X-Box controllers) the Rumble control will appear. This can be used to test the rumble functionality of the device. The top bar is "Right Rumble" and the bottom bar is "Left Rumble". Clicking and holding anywhere along the bar will activate the rumble proportionally (left is no rumble = 0, right is full rumble = 1). This is a control only and will not indicate the Rumble value set in robot code.

Re-Arranging and Locking Devices



The Driver Station has the capability of “locking” a USB device into a specific slot. This is done automatically if the device is dragged to a new position and can also be triggered by double clicking on the device. “Locked” devices will show up with an underline under the device. A locked device will reserve it’s slot even when the device is not connected to the computer (shown as grayed out and underlined). Devices can be unlocked (and unconnected devices removed) by double clicking on the entry.

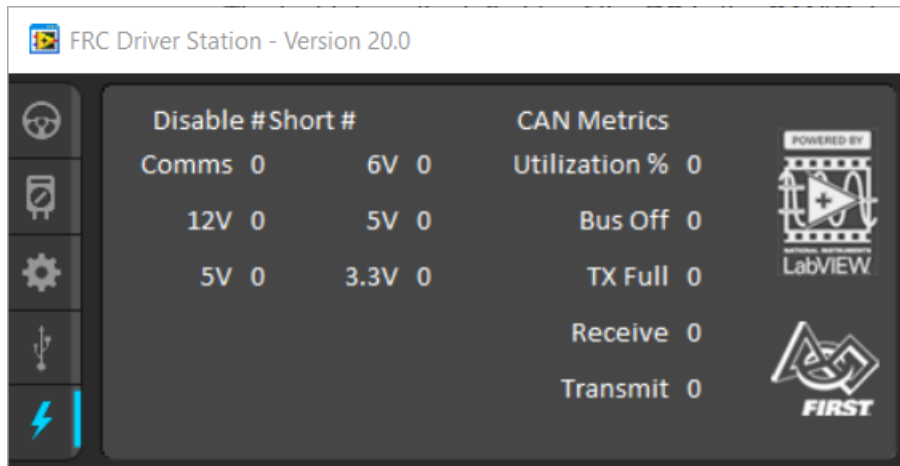
Note: If you have two or more of the same device, they should maintain their position as long as all devices remain plugged into the computer in the same ports they were locked in. If you switch the ports of two identical devices the lock should follow the port, not the device. If you re-arrange the ports (take one device and plug it into a new port instead of swapping) the behavior is not determinate (the devices may swap slots). If you unplug one or more of the set of devices, the positions of the others may move, they should return to the proper locked slots when all devices are reconnected.

Example: The image above shows 4 devices:

- A Locked “Logitech Attack 3” joystick. This device will stay in this position unless dragged somewhere else or unlocked
- An unlocked “Logitech Extreme 3D” joystick
- An unlocked “Gamepad F310 (Controller)” which is a Logitech F310 gamepad
- A Locked, but disconnected “MadCatz GamePad (Controller)” which is a MadCatz Xbox 360 Controller

In this example, unplugging the Logitech Extreme 3D joystick will result in the F310 Gamepad moving up to slot 1. Plugging in the MadCatz Gamepad (even if the devices in Slots 1 and 2 are removed and those slots are empty) will result in it occupying Slot 3.

20.2.8 CAN/Power Tab

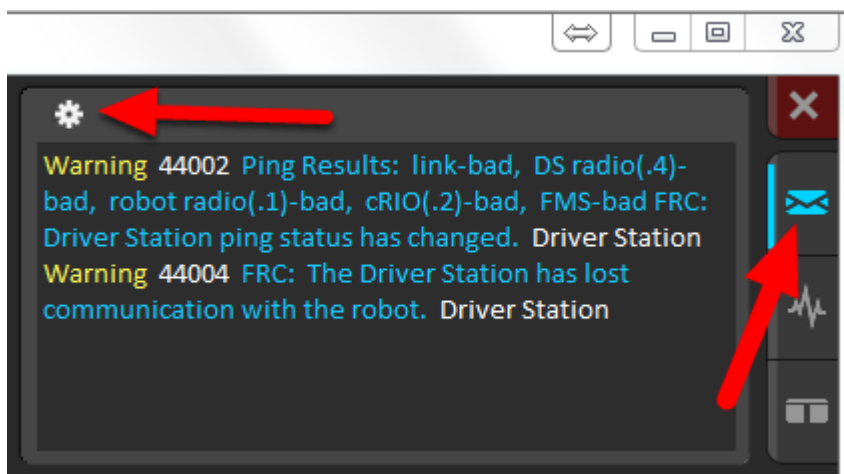


The last tab on the left side of the DS is the CAN/Robot Power Tab. This tab contains information about the power status of the roboRIO and the status of the CAN bus:

1. Comms Faults - Indicates the number of Comms faults that have occurred since the DS has been connected
2. 12V Faults - Indicates the number of input power faults (Brownouts) that have occurred since the DS has been connected
3. 6V/5V/3.3V Faults - Indicates the number of faults (typically cause by short circuits) that have occurred on the User Voltage Rails since the DS has been connected
4. CAN Bus Utilization - Indicates the percentage utilization of the CAN bus
5. CAN faults - Indicates the counts of each of the 4 types of CAN faults since the DS has been connected

If a fault is detected, the indicator for this tab (shown in blue in the image above) will turn red.

20.2.9 Messages Tab



The Messages tab displays diagnostic messages from the DS, WPILib, User Code, and/or the roboRIO. The messages are filtered by severity. By default, only Errors are displayed.

To access settings for the Messages tab, click the Gear icon. This will display a menu that will allow you to select the detail level (Errors, Errors+Warnings or Errors+Warnings+Prints), Clear the box, launch a larger Console window for viewing messages, or launch the DS Log Viewer.

20.2.10 Charts Tab



The Charts tab plots and displays advanced indicators of robot status to help teams diagnose robot issues:

1. The top graph charts trip time in milliseconds in green (against the axis on the right) and lost packets per second in orange (against the axis on the left)
2. The bottom graph plots battery voltage in yellow (against the axis on the left), roboRIO CPU in red (against the axis on the right), DS Requested mode as a continuous line on the bottom of the chart and robot mode as a discontinuous line above it.
3. This key shows the colors used for the DS Requested and Robot Reported modes in the bottom chart.
4. Chart scale - These controls change the time scale of the DS Charts
5. This button launches the *DS Log File Viewer*

The DS Requested mode is the mode that the Driver Station is commanding the robot to be in. The Robot Reported mode is what code is actually running based on reporting methods contained in the coding frameworks for each language.

20.2.11 Both Tab

The last tab on the right side is the Both tab which displays Messages and Charts side by side

20.2.12 Driver Station Key Shortcuts

- *F1* - Force a Joystick refresh.

- `[+] + \` - Enable the robot (the 3 keys above Enter on most keyboards)
- *Enter* - Disable the Robot
- *Space* - Emergency Stop the robot. After an emergency stop is triggered the roboRIO will need to be rebooted before the robot can be enabled again.

Note: Space bar will E-Stop the robot regardless of if the Driver Station window has focus or not

20.3 Programming Radios for FMS Offseason

When using the FMS Offseason software, the typical networking setup is to use a single access point with a single SSID and WPA key. This means that the radios should all be programmed to connect to this network, but with different IPs for each team. The Team version of the FRC Bridge Configuration Utility has an FMS-Lite mode that can be used to do this configuration.

Before you begin using the software:

1. Disable WiFi connections on your computer, as it may prevent the configuration utility from properly communicating with the bridge
2. Make sure no devices are connected to your computer via ethernet, other than the wireless bridge.

20.3.1 Pre-Requisites

Note: Even though WPILib uses Java 11, the FRC Radio Configuration Utility requires Java 8.

The FRC Radio Configuration Utility requires the Java Runtime Engine (JRE). If you do not have Java installed, you can download the JRE from [here](#).

The FRC Radio Configuration Utility requires Administrator privileges to configure the network settings on your machine. The program should request the necessary privileges automatically (may require a password if run from a non-Administrator account), but if you are having trouble try running it from an Administrator account.

20.3.2 Application Notes

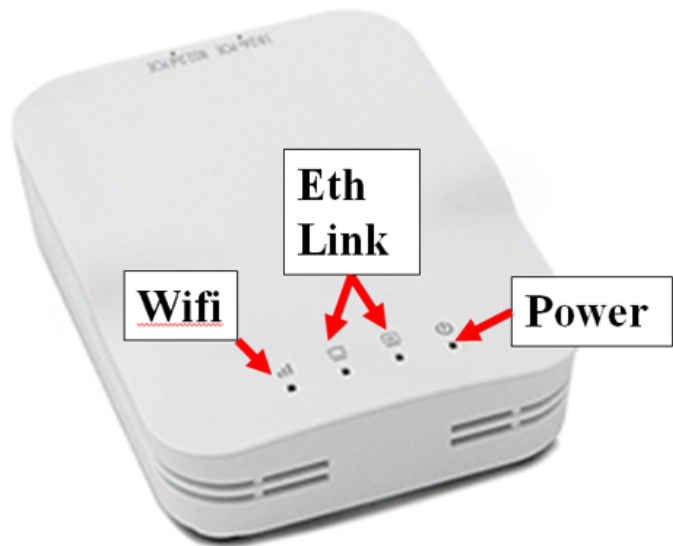
The Radio Kiosk will program the radio to enforce the 4 Mbps bandwidth limit on traffic exiting the radio over the wireless interface. In the home configuration (AP mode) this is a total, not a per client limit. This means that streaming video to multiple clients is not recommended.

The Kiosk has been tested on Windows 7, 8, and 10. It may work on other operating systems, but has not been tested.

Programmed Configuration

Power	
Blue	On or Powering up
Blue Blinking	Powering up
Eth Link	
Blue	Link Up
Blue Blinking	Traffic present
WiFi	
Red	Bridge Mode, unlinked
Yellow\Orange	Bridge Mode, Linked
Green	AP Mode
Off	Unprogrammed

WiFi light only works after radio has been power cycled.



The Radio Configuration Utility programs a number of configuration settings into the radio when run. These settings apply to the radio in all modes (including at events). These include:

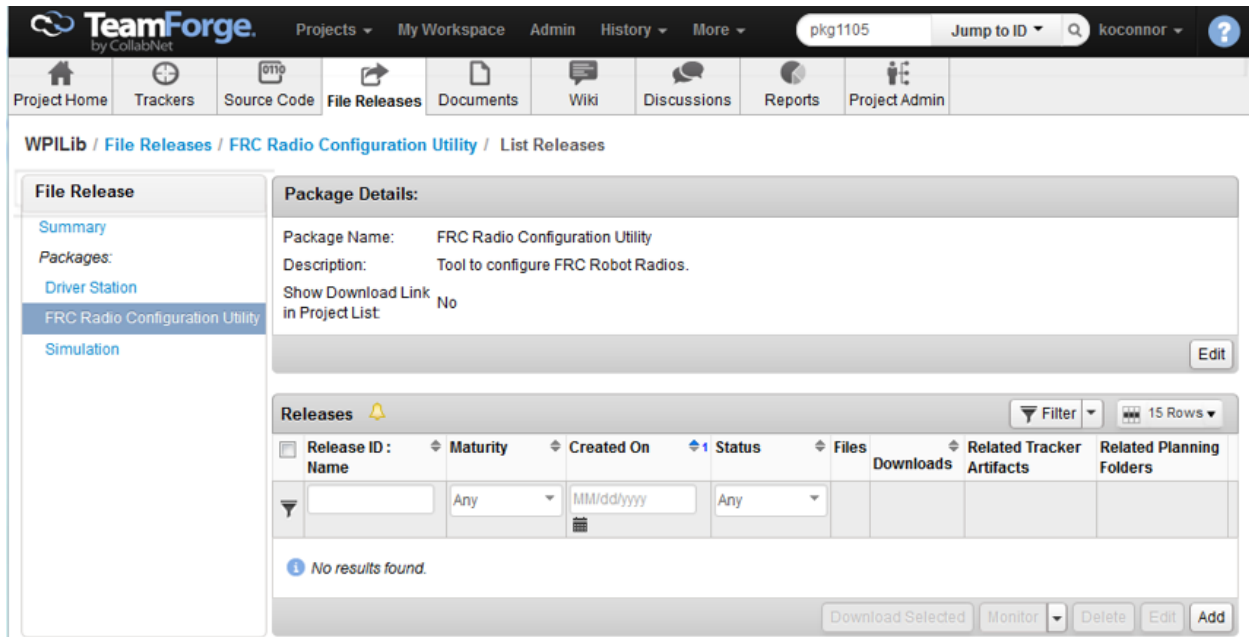
- Set a static IP of 10.TE.AM.1
- Set an alternate IP on the wired side of 192.168.1.1 for future programming
- Bridge the wired ports so they may be used interchangeably
- The LED configuration noted in the graphic above
- 4Mb/s bandwidth limit on the outbound side of the wireless interface
- QoS rules for internal packet prioritization (affects internal buffer and which packets to discard if bandwidth limit is reached). These rules are Robot Control and Status (UDP 1110, 1115, 1150) >> Robot TCP & Network Tables (TCP 1735, 1740) >> Bulk (All other traffic).

When programmed with the team version of the Radio Configuration - Utility, the user accounts will be left at (or set to) the firmware - defaults **for the DAPs only**:

- Username: root
- Password: root

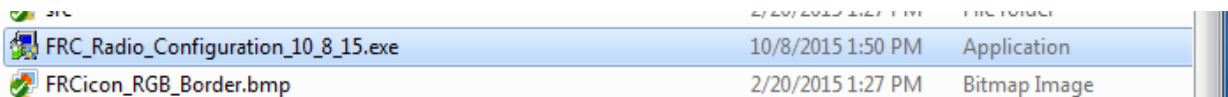
Note: It is not recommended to modify the configuration manually

20.3.3 Download the software



Download the latest FRC Radio Configuration Utility Installer from the [WPILib project File Releases](#).

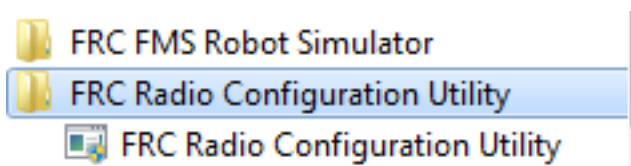
20.3.4 Install the software



Double click on FRC_Radio_Configuration_MM_DD_YY.exe to launch the installer. Follow the prompts to complete the installation.

Part of the installation prompts will include installing WinPCap if it is not already present. The WinPCap installer contains a checkbox (checked by default) to start the WinPCap driver on boot. You should leave this box checked.

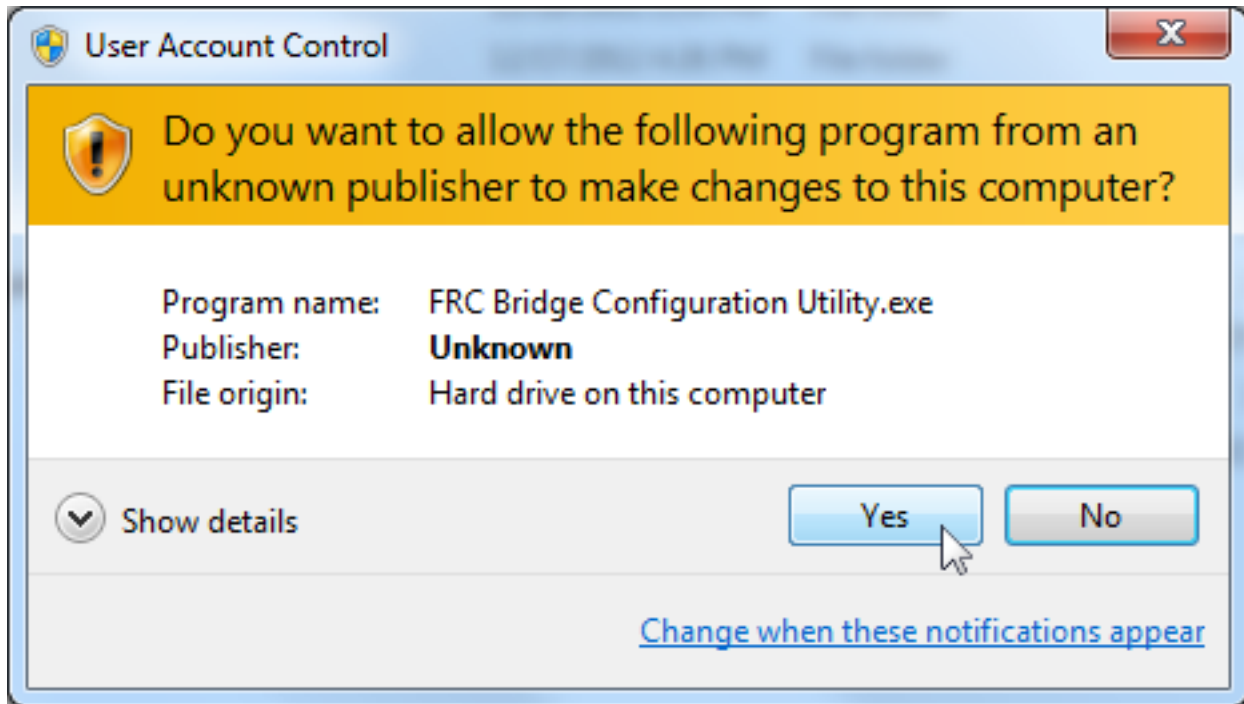
20.3.5 Launch the software



Use the Start menu or desktop shortcut to launch the program.

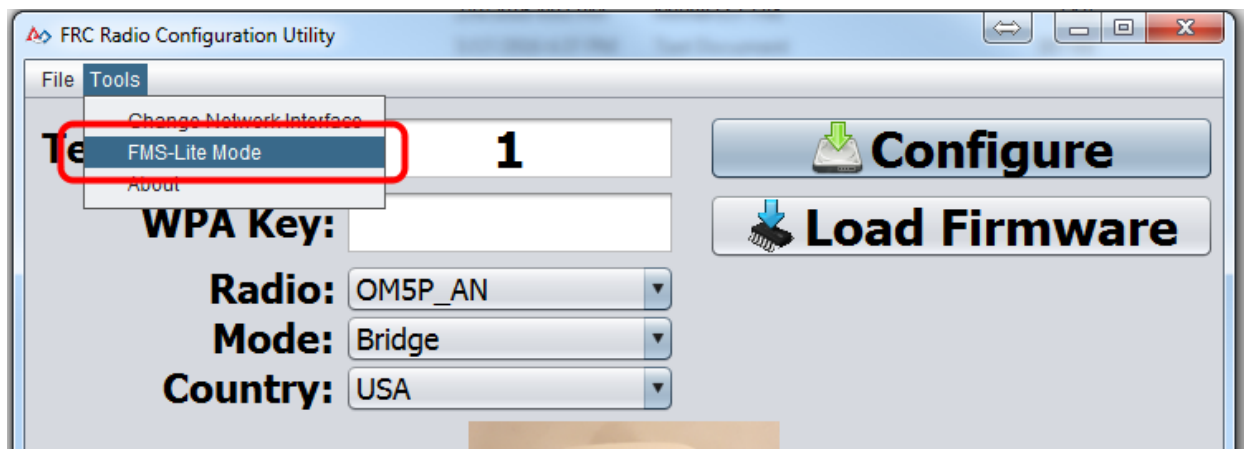
Note: If you need to locate the program it is installed to C:/Program Files (x86)/FRC Radio Configuration Utility. For 32-bit machines the path is C:/Program Files/FRC

20.3.6 Allow the program to make changes, if prompted



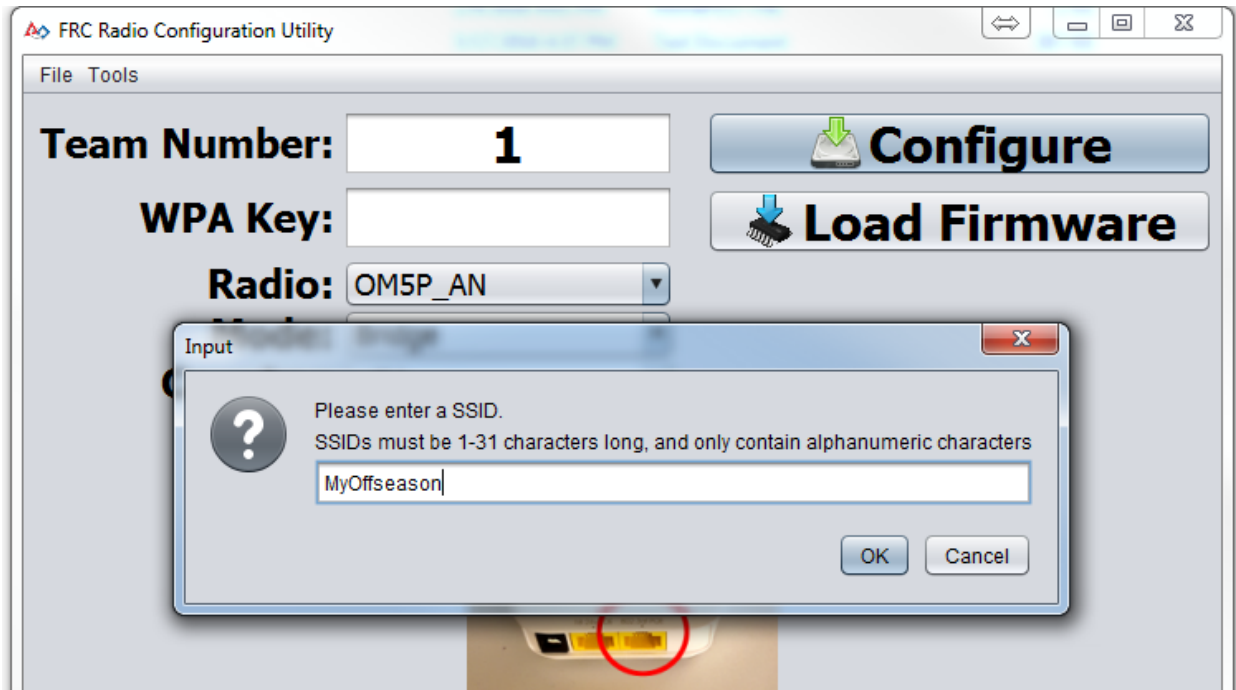
If the your computer is running Windows Vista or Windows 7, a prompt may appear about allowing the configuration utility to make changes to the computer. Click "Yes" if the prompt appears.

20.3.7 Enter FMS-Lite Mode



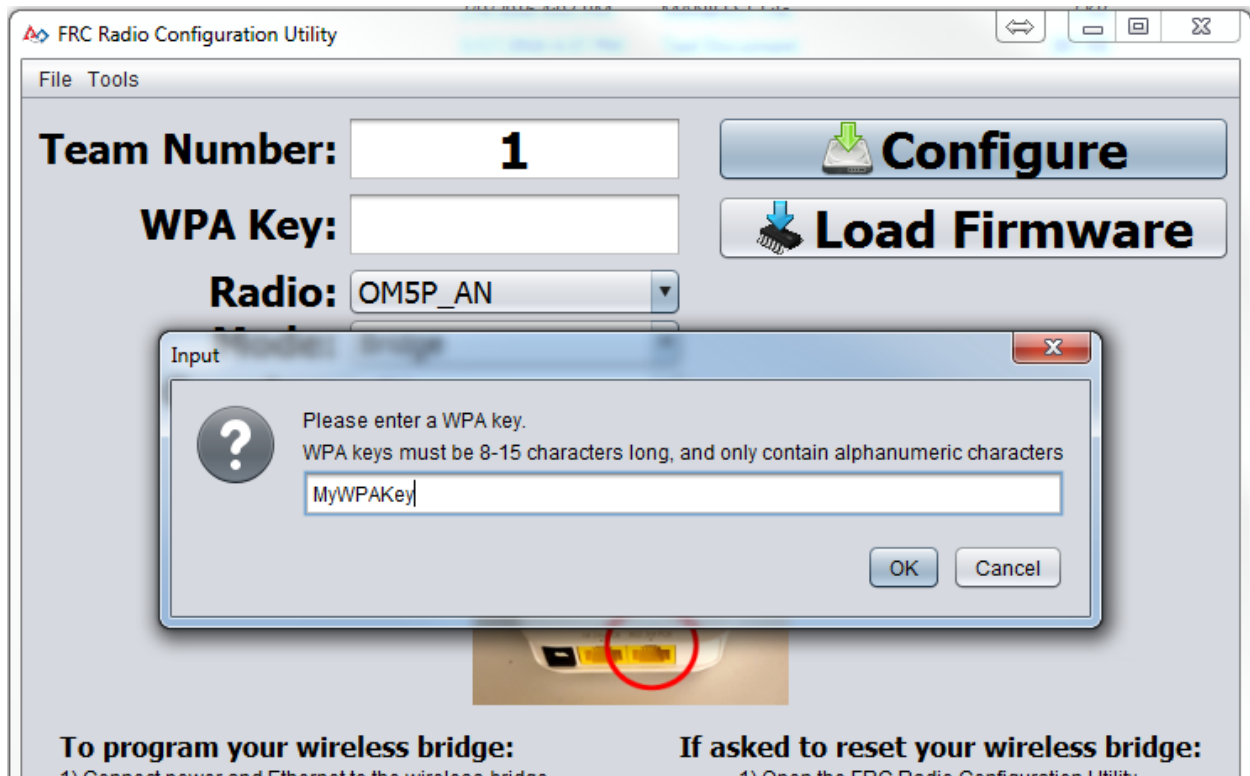
Click Tools -> FMS-Lite Mode to enter FMS-Lite Mode.

20.3.8 Enter SSID



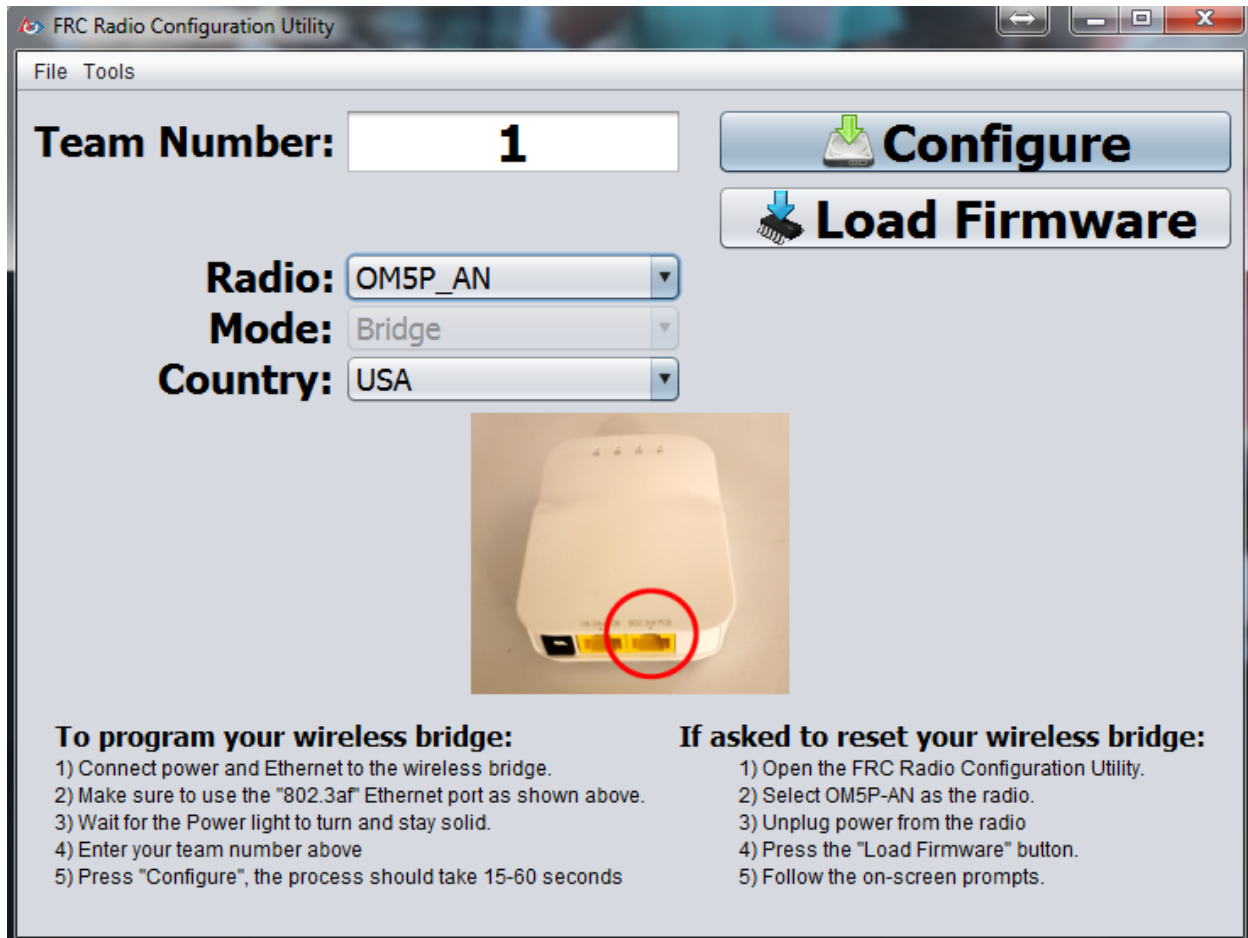
Enter the SSID (name) of your wireless network in the box and click OK.

20.3.9 Enter WPA Key



Enter the WPA key for your network in the box and click OK. Leave the box blank if you are using an unsecured network.

20.3.10 Program Radios



The Kiosk is now ready to program any number of radios to connect to the network entered. To program each radio, connect the radio to the Kiosk, set the Team Number in the box, and click Configure.

The kiosk will program OpenMesh, D-Link Rev A or D-Link Rev B radios to work on an offseason FMS network by selecting the appropriate option from the "Radio" dropdown.

Note: Bandwidth limitations and QoS will not be configured on the D-Link radios in this mode.

20.3.11 Changing SSID or Key

If you enter something incorrectly or need to change the SSID or WPA Key, go to the Tools menu and click FMS-Lite Mode to take the kiosk out of FMS-Lite Mode. When you click again to put the Kiosk back in FMS-Lite Mode, you will be re-prompted for the SSID and Key.

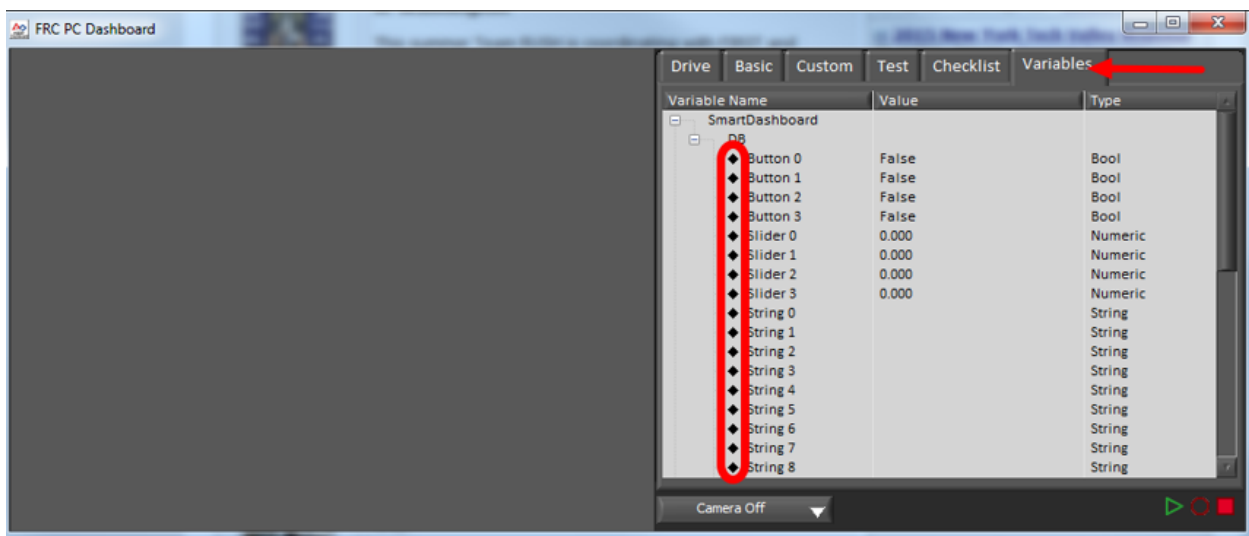
20.4 Troubleshooting Dashboard Connectivity

We have received a number of reports of Dashboard connectivity issues from events. This document will help explain how to recognize if the Dashboard is not connected to your robot, steps to troubleshoot this condition and a code modification you can make.

20.4.1 LabVIEW Dashboard

This section discusses connectivity between the robot and LabVIEW dashboard

Recognizing LabVIEW Dashboard Connectivity



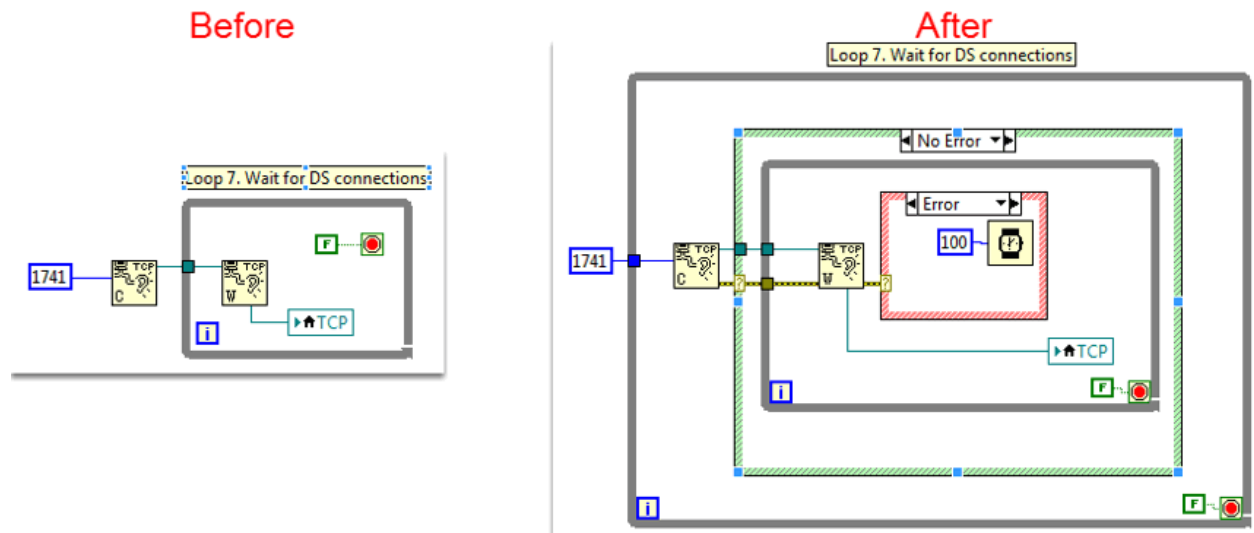
If you have an indicator on your dashboard that you expect to be changing it may be fairly trivial to recognize if the Dashboard is connected. If not, there is a way to check without making any changes to your robot code. On the Variables tab of the Dashboard, the variables are shown with a black diamond when they are not synced with the robot. Once the Dashboard connects to the robot and these variables are synced, the diamond will disappear.

Troubleshooting LabVIEW Dashboard Connectivity

If the Dashboard does not connect to the Robot (after the Driver Station has connected to the robot) the recommended troubleshooting steps are:

1. Close the Driver Station and Dashboard, then re-open the Driver Station (which should launch the Dashboard).
2. If that doesn't work, restart the Robot Code using the Restart Robot Code button on the Diagnostics tab of the Driver Station

Improving Reliability of a Custom Dashboard

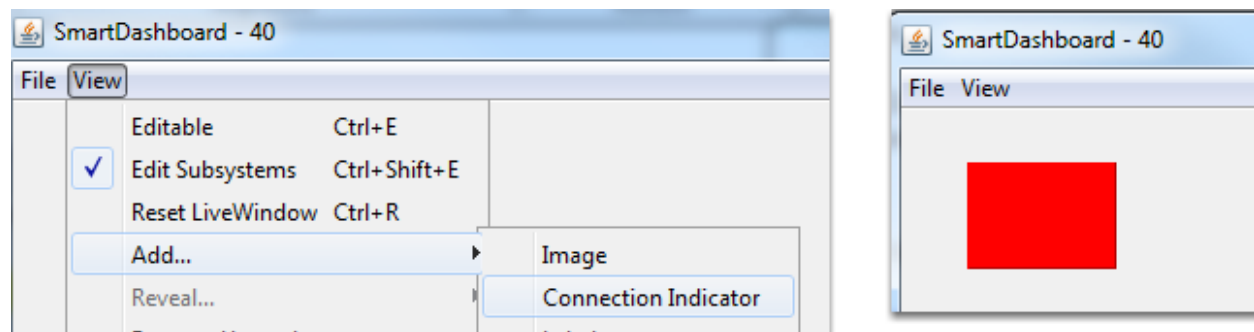


If you have created a custom LabVIEW dashboard there is a tweak you can make to the code to improve reliability of the initial connection. Locate the loop labeled Loop 7 in the Dashboard Main VI. Modify the loop according to the image above by adding a loop around the listener, 2 case statements, a Wait block and error wiring.

20.4.2 SmartDashboard

This section discusses connectivity between the robot and Java SmartDashboard

Recognizing SmartDashboard Connectivity



The typical way to recognize connectivity with the Java SmartDashboard is to add a Connection Indicator widget and to make sure your code is writing at least one key during initialization or disabled to trigger the connection indicator. The connection indicator can be moved or re-sized if the Editable checkbox is checked.

Troubleshooting SmartDashboard Connectivity

If the Dashboard does not connect to the Robot (after the Driver Station has connected to the robot) the recommended troubleshooting steps are:

1. Restart the SmartDashboard (there is no need to restart the Driver Station software for the Java SmartDashboard)
2. If that doesn't work, restart the Robot Code using the Restart Robot Code button on the Diagnostics tab of the Driver Station
3. If it still doesn't connect verify that the Team Number is set properly in the Dashboard and that your Robot Code writes a SmartDashboard value during initialization or disabled

20.5 Driver Station Best Practices

This document was created by Steve Peterson, with contributions from Juan Chong, James Cole-Henry, Rick Kosbab, Greg McKaskle, Chris Picone, Chris Roadfeldt, Joe Ross, and Ryan Sjostrand. The original post and follow-up posts can be found here: <https://www.chiefdelphi.com/t/paper-driver-station-best-practices/164429> and a mirror of the document can be found [here](#).

Want to ensure the driver station isn't a stopper for your team at the FIRST Robotics Competition (FRC) field? Building and configuring a solid driver station laptop is an easy project for the time between stop build day and your competition. Read on to find lessons learned by many teams over thousands of matches.

20.5.1 Prior To Departing For The Competition

1. Dedicate a laptop to be used solely as a driver station. Many teams do. A dedicated machine allows you manage the configuration for one goal – being ready to compete at the field. Dedicated means no other software except the FRC-provided Driver Station software and associated Dashboard installed or running.
2. Use a business-class laptop for your driver station. Why? They're much more durable than the \$300 Black Friday special at Best Buy. They'll survive being banged around at the competition. Business-class laptops have higher quality device drivers, and the drivers are maintained for a longer period than consumer laptops. This makes your investment last longer. Lenovo ThinkPad T series and Dell Latitude are two popular business-class brands you'll commonly see at competitions. There are thousands for sale every day on eBay. The laptop provided in recent rookie kits is a good entry level machine. Teams often graduate from it to bigger displays as they do more with vision and dashboards.
3. Consider used laptops rather than new. The FRC Driver Station and dashboard software uses very few system resources, so you don't need to buy a new laptop – instead, buy a cheap 4-5 year old used one. You might even get one donated by a used computer store in your area.
4. Laptop recommended features
 - a. RAM – 2GB of RAM is minimum, if you have a SSD.
 - b. A display size of 13" or greater, with minimum resolution of 1440x1050.

- c. Ports
 - i. A built-in Ethernet port is highly preferred. Ensure that it's a full-sized port. The hinged Ethernet ports don't hold up to repeated use.
 - ii. Use an Ethernet port saver to make your Ethernet connection. This extends the life of the port on the laptop. This is particularly important if you have a consumer-grade laptop with a hinged Ethernet port.
 - iii. If the Ethernet port on your laptop is dodgy, either replace the laptop (recommended) or buy a USB Ethernet dongle from a reputable brand. Many teams find that USB Ethernet is less reliable than built-in Ethernet, primarily due to cheap hardware and bad drivers. The dongles given to rookies in the KOP have a reputation for working well.
 - iv. 2 USB ports minimum
- d. A keyboard. It's hard to quickly do troubleshooting on touch-only computers at the field.
- e. A solid-state disk (SSD). If the laptop has a rotating disk, spend \$50 and replace it with a SSD.
- f. Updated to the current release of Windows 10. Being the most common OS now seen at competitions, bugs are more likely to be found and fixed for Windows 10 than on older Windows versions.
- 5. Install all Windows updates a week before the competition. This allows you time to ensure the updates will not interfere with driver station functions. To do so, open the Windows Update settings page and see that you're up-to-date. Install pending updates if not. Reboot and check again to make sure you're up to date.
- 6. Change "Active Hours" for Windows Updates to prevent updates from installing during competition hours. Navigate to Start -> Settings -> Update & Security -> Windows Update, then select Change active hours. If you're traveling to a competition, take time zone differences into account. This will help ensure your driver station does not reboot or fail due to update installing on the field.
- 7. Remove any 3rd party antivirus or antimalware software. Instead, use Windows Defender on Windows 10. Since you're only connecting to the internet for Windows and FRC software updating, the risk is low. Only install software on your driver station that's needed for driving. Your goal here is to eliminate variables that might interfere with proper operation. Remove any unneeded preinstalled software ("crapware") that came with the machine. Don't use the laptop as your Steam machine for gaming back at the hotel the night before the event. Many teams go as far as having a separate programming laptop.
- 8. Avoid managed Windows 10 installations from the school's IT department. These deployments are built for the school environment and often come with unwanted software that interferes with your robot's operation.
- 9. Laptop battery / power
 - a. Turn off Put the computer to sleep in your power plan for both battery and powered operation.
 - b. Turn off USB Selective Suspend:
 - i. Right click on the battery/charging icon in the tray, then select Power Options.
 - ii. Edit the plan settings of your power plan.

- iii. Click the Change advanced power settings link.
 - iv. Scroll down in the advanced settings and disable the USB selective suspend setting for both Battery and Plugged in.
 - c. Ensure the laptop battery can hold a charge for at least an hour after making the changes above. This allows plenty of time for the robot and drive team to go through the queue and reach the alliance station without mains power.
10. Bring a trusted USB and Ethernet cable for use connecting to the roboRIO.
 11. Add retention/strain relief to prevent your joystick/gamepad controllers from falling on the floor and/or yanking on the USB ports. This helps prevent issues with intermittent controller connections.
 12. The Windows user account you use to drive must be a member of the Administrator group.

20.5.2 At The Competition

1. Turn off Windows firewall using these instructions.
2. Turn off the Wi-Fi adapter, either using the dedicated hardware Wi-Fi switch or by disabling it in the Adapter Settings control panel.
3. Charge the driver station when it's in the pit.
4. Remove login passwords or ensure everyone on the drive team knows the password. You'd be surprised at how often drivers arrive at the field without knowing the password for the laptop.
5. Ensure your LabView code is deployed permanently and set to "run as startup", using the instructions in the LabView Tutorial. If you must deploy code every time you turn the robot on, you're doing it wrong.
6. Limit web browsing to FRC related web sites. This minimizes the chance of getting malware during the competition.
7. Don't plan on using internet access to do software updates. There likely won't be any in the venue, and hotel Wi-Fi varies widely in quality. If you do need updates, contact a Control System Advisor in the pit.

20.5.3 Before Each Match

1. Make sure the laptop is on and logged in prior to the end of the match before yours.
2. Close programs that aren't needed during the match - e.g., Eclipse or LabView - when you are competing.
3. Bring your laptop charger to the field. Power is provided for you in each player station.
4. Fasten your laptop with hook-and-loop tape to the player station shelf. You never know when your alliance partner will have an autonomous programming issue and blast the wall.
5. Ensure joysticks and controllers are assigned to the correct USB ports.
 - a. In the USB tab in the FRC Driver Station software, drag and drop to assign joysticks as needed.

- b. Use the rescan button (F1) if joysticks / controllers do not appear green
- c. Use the rescan button (F1) during competition if joystick or controllers become unplugged and then are plugged back in or otherwise turn gray during competition.

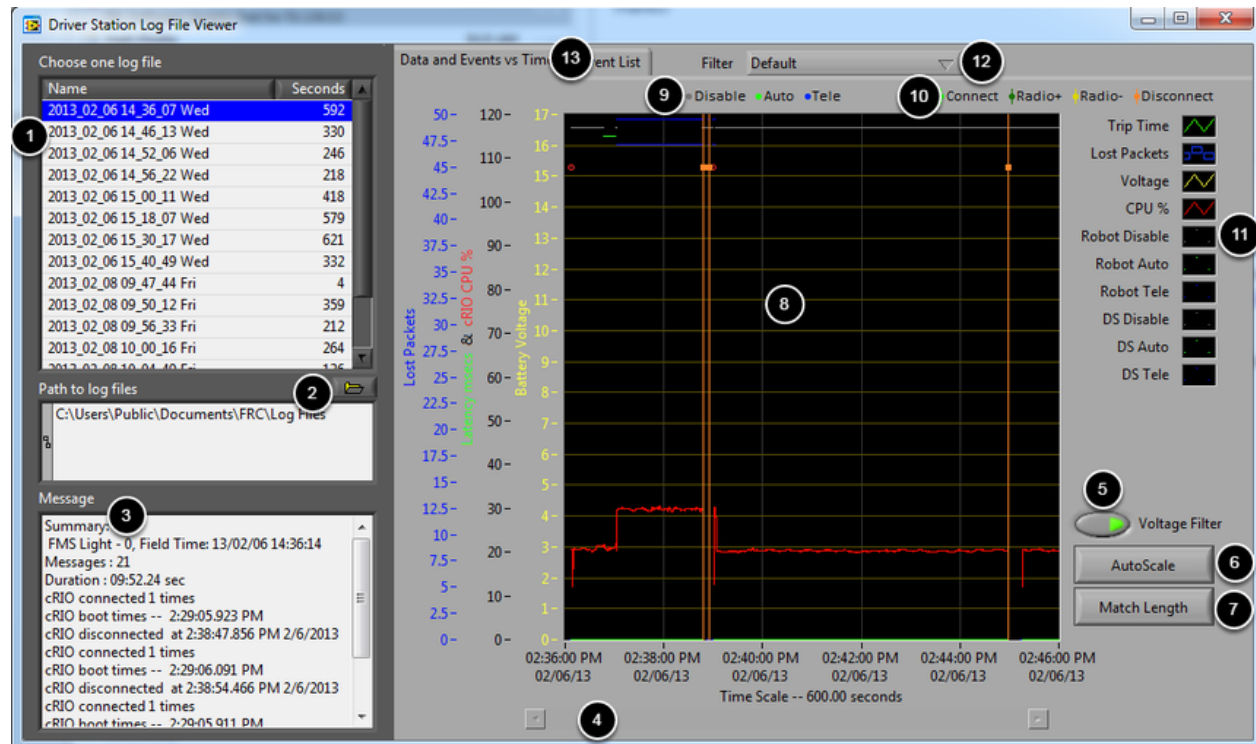
20.6 Driver Station Log File Viewer

In an effort to provide information to aid in debugging, the FRC Driver Station creates log files of important diagnostic data while running. These logs can be reviewed later using the FRC Driver Station Log Viewer. The Log Viewer can be found via the shortcut installed in the Start menu or in the FRC Driver Station folder in Program Files.

20.6.1 Event Logs

The Driver Station logs all messages sent to the Messages box on the Diagnostics tab (not the User Messages box on the Operation tab) into a new Event Log file. When viewing Log Files with the Driver Station Log File Viewer, the Event Log and DSLog files are overlaid in a single display.

20.6.2 Log Viewer UI

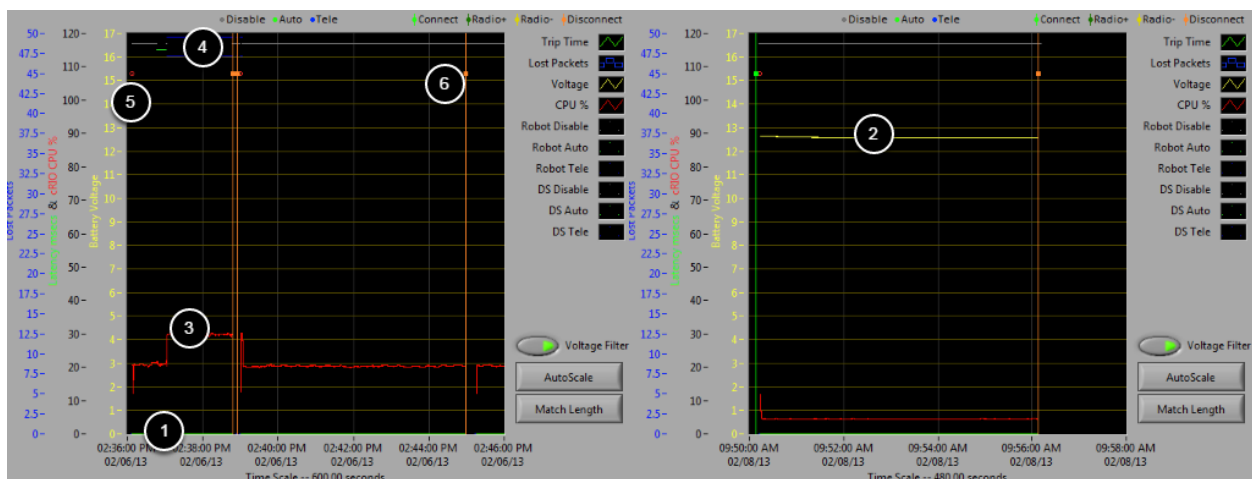


The Log Viewer contains a number of controls and displays to aid in the analysis of the Driver Station log files:

1. File Selection Box - This window displays all available log files in the currently selected folder. Click on a log file in the list to select it.

2. Path to Log Files - This box displays the current folder the viewer is looking in for log files. This defaults to the folder that the Driver Station stores log files in. Click the folder icon to browse to a different location.
3. Message Box - This box displays a summary of all messages from the Event Log. When hovering over an event on the graph this box changes to display the information for that event.
4. Scroll Bar - When the graph is zoomed in, this scroll bar allows for horizontal scrolling of the graph.
5. Voltage Filter - This control turns the Voltage Filter on and off (defaults to on). The Voltage Filter filters out data such as CPU %, robot mode and trip time when no Battery Voltage is received (indicating that the DS is no in communication with the roboRIO).
6. AutoScale - This button zooms the graph out to show all data in the log.
7. Match Length - This button scales the graph to approximately the length of an FRC match (2 minutes and 30 seconds shown). It does not automatically locate the start of the match, you will have to scroll using the scroll bar to locate the beginning of the Autonomous mode.
8. Graph - This display shows graph data from the DS Log file (voltage, trip time, roboRIO CPU%, Lost Packets, and robot mode) as well as overlaid event data (shown as dots on the graph with select events showing as vertical lines across the entire graph). Hovering over event markers on the graph displays information about the event in the Messages window in the bottom left of the screen.
9. Robot Mode Key - Key for the Robot Mode displayed at the top of the screen
10. Major event key - Key for the major events, displayed as vertical lines on the graph
11. Graph key - Key for the graph data
12. Filter Control - Drop-down to select the filter mode (filter modes explained below)
13. Tab Control - Control to switch between the Graph (Data and Events vs. Time) and Event List displays.

20.6.3 Using the Graph display



The Graph Display contains the following information:

1. Graphs of Trip Time in ms (green line) and Lost Packets per second (displayed as blue vertical bars). In these example images Trip Time is a flat green line at the bottom of the graph and there are no lost packets
 2. Graph of Battery voltage displayed as a yellow line.
 3. Graph of roboRIO CPU % as a red line
 4. Graph of robot mode and DS mode. The top set of the display shows the mode commanded by the Driver Station. The bottom set shows the mode reported by the robot code. In this example the robot is not reporting it's mode during the disabled and autonomous modes, but is reported during Teleop.
 5. Event markers will be displayed on the graph indicating the time the event occurred. Errors will display in red; warnings will display in yellow. Hovering over an event marker will display information about the event in the Messages box at the bottom left of the screen.
 6. Major events are shown as vertical lines across the graph display.
- To zoom in on a portion of the graph, click and drag around the desired viewing area. You can only zoom the time axis, you cannot zoom vertically.

20.6.4 Event List

DS Time	Event Message Text
2:36:07.288 PM	WARNING <Code> 44007 occurred at FRC_NetworkCommunications <secondsSinceReboot> 421.365 Warning <Code> 44001 occurred at No Change to Network Configuration: "Local Area Connection" <noNIC> FRC: Time since robot boot. Driver Station <time> 2/6/2013 2:36:07 PM<unique#> 3 ERROR <Code> -44009 occurred at Driver Station <time> 2/6/2013 2:36:06 PM<unique#> 2 FRC: A joystick was disconnected while the robot was enabled. Warning <Code> 44006 occurred at Driver Station <time> 2/6/2013 2:36:06 PM<unique#> 1 FRC: Custom I/O is not enabled or is not connected to the driver station.
2:36:07.328 PM	FMS Connected: FMS Light - 0, Field Time: 13/02/06 14:36:14
2:36:10.441 PM	WARNING <Code> 44008 occurred at FRC_NetworkCommunications <radioLostEvents> 173.563 <radioSec> FRC: Robot radio detection times.
2:37:01.461 PM	Watchdog Expiration: System 1, User 0
2:38:47.856 PM	Warning <Code> 44004 occurred at Driver Station <time> 2/6/2013 2:38:47 PM<unique#> 4 FRC: The Driver Station has lost communication with the robot.
2:38:49.356 PM	Warning <Code> 44002 occurred at Ping Results: link-GOOD, DS radio(4)-GOOD, robot radio(1)-GOOD, <time> 2/6/2013 2:38:49 PM<unique#> 5 FRC: Driver Station ping status has changed.
2:38:53.460 PM	WARNING <Code> 44007 occurred at FRC_NetworkCommunications <secondsSinceReboot> 587.369 FRC: Time since robot boot.
2:38:54.466 PM	Warning <Code> 44004 occurred at Driver Station <time> 2/6/2013 2:38:53 PM<unique#> 6 FRC: The Driver Station has lost communication with the robot.
2:38:55.468 PM	Warning <Code> 44002 occurred at Ping Results: link-GOOD, DS radio(4)-GOOD, robot radio(1)-GOOD, <time> 2/6/2013 2:38:55 PM<unique#> 7 FRC: Driver Station ping status has changed.
2:38:59.278 PM	WARNING <Code> 44008 occurred at FRC_NetworkCommunications <radioLostEvents> 339.065 <radioSec> FRC: Robot radio detection times. WARNING <Code> 44007 occurred at FRC_NetworkCommunications <secondsSinceReboot> 593.367

The Event List tab displays a list of events (warnings and errors) recorded by the Driver Station. The events and detail displayed are determined by the currently active filter (images shows “All Events, All Info” filter active).

20.6.5 Filters

Three filters are currently available in the Log Viewer:

1. Default: This filter filters out many of the errors and warnings produced by the Driver Station. This filter is useful for identifying errors thrown by the code on the Robot.
2. All Events and Time: This filter shows all events and the time they occurred
3. All Events, All Info: This filter shows all events and all recorded info. At this time the primary difference between this filter and “All Events and Time” is that this option shows the “unique” designator for the first occurrence of a particular message.

20.6.6 Identifying Logs from Matches

3:19:30.893 PM	FMS Connected: Practice - 1, Field Time: 13/02/06 15:19:37
----------------	--

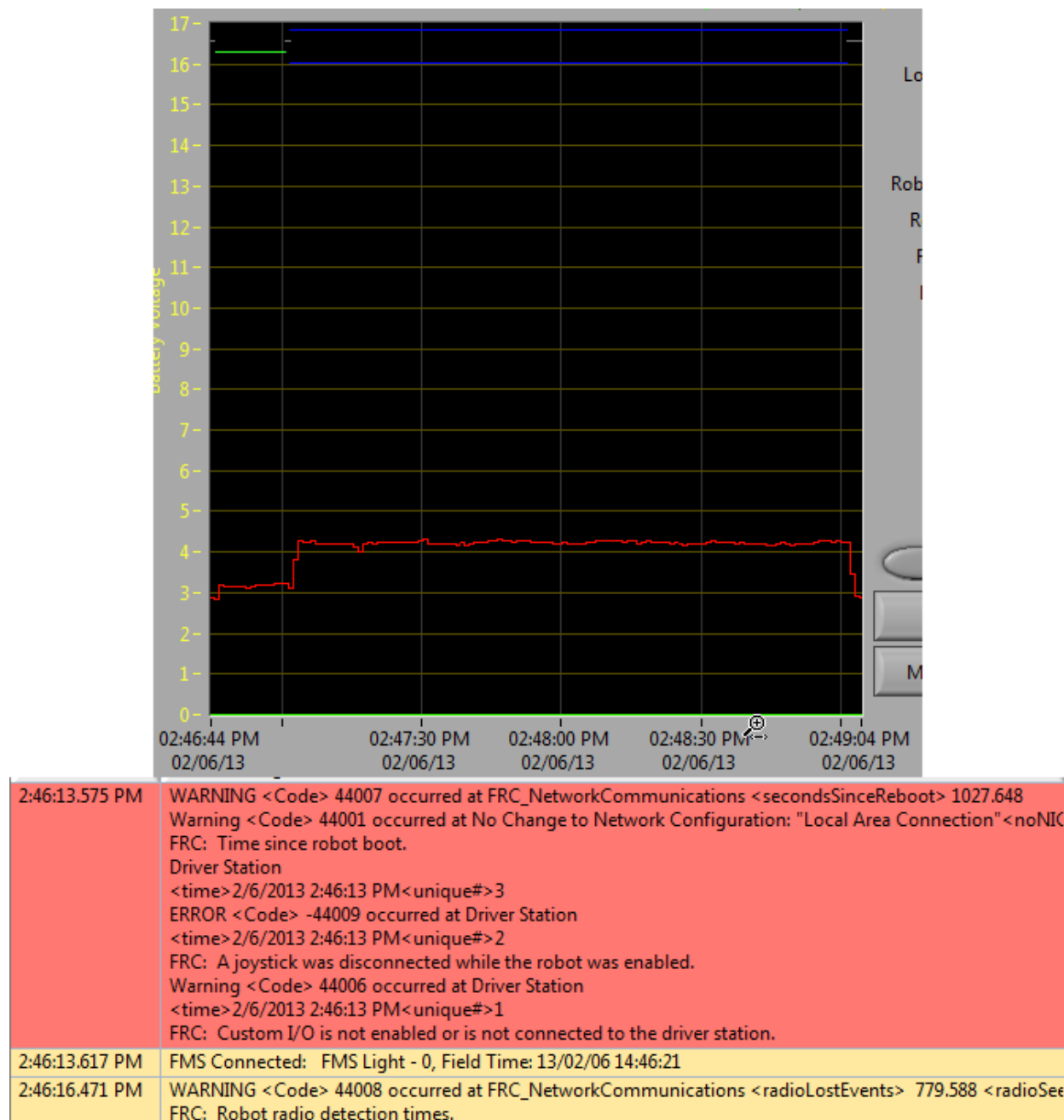
A common task when working with the Driver Station Logs is to identify which logs came from competition matches. Logs which were taken during a match can now be identified using the FMS Connected event which will display the match type (Practice, Qualification or Elimination), match number, and the current time according to the FMS server. In this example, you can see that the FMS server time and the time of the Driver Station computer are fairly close, approximately 7 seconds apart.

20.6.7 Identifying Common Connection Failures with the Log Viewer

When diagnosing robot issues, there is no substitute for thorough knowledge of the system and a methodical debugging approach. If you need assistance diagnosing a connection problem at your events it is strongly recommended to seek assistance from your FTA and/or CSA. The goal of this section is to familiarize teams with how some common failures can manifest themselves in the DS Log files. Please note that depending on a variety of conditions a particular failure show slightly differently in a log file.

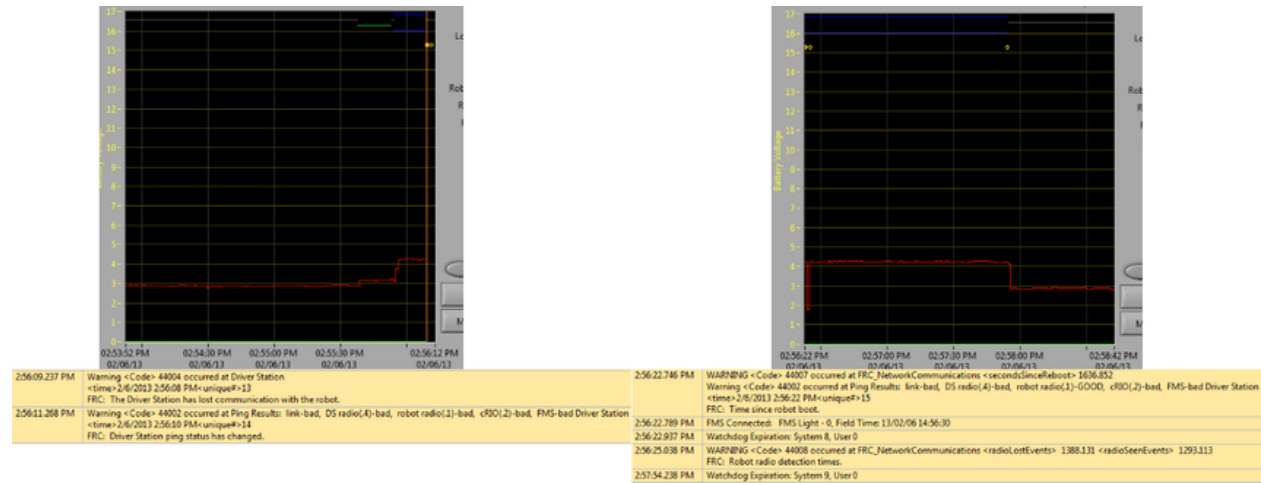
Note that all log files shown in this section have been scaled to match length using the Match Length button and then scrolling to the beginning of the autonomous mode. Also, many of the logs do not contain battery voltage information, the platform used for log capture was not properly wired for reporting the battery voltage.

“Normal” Log



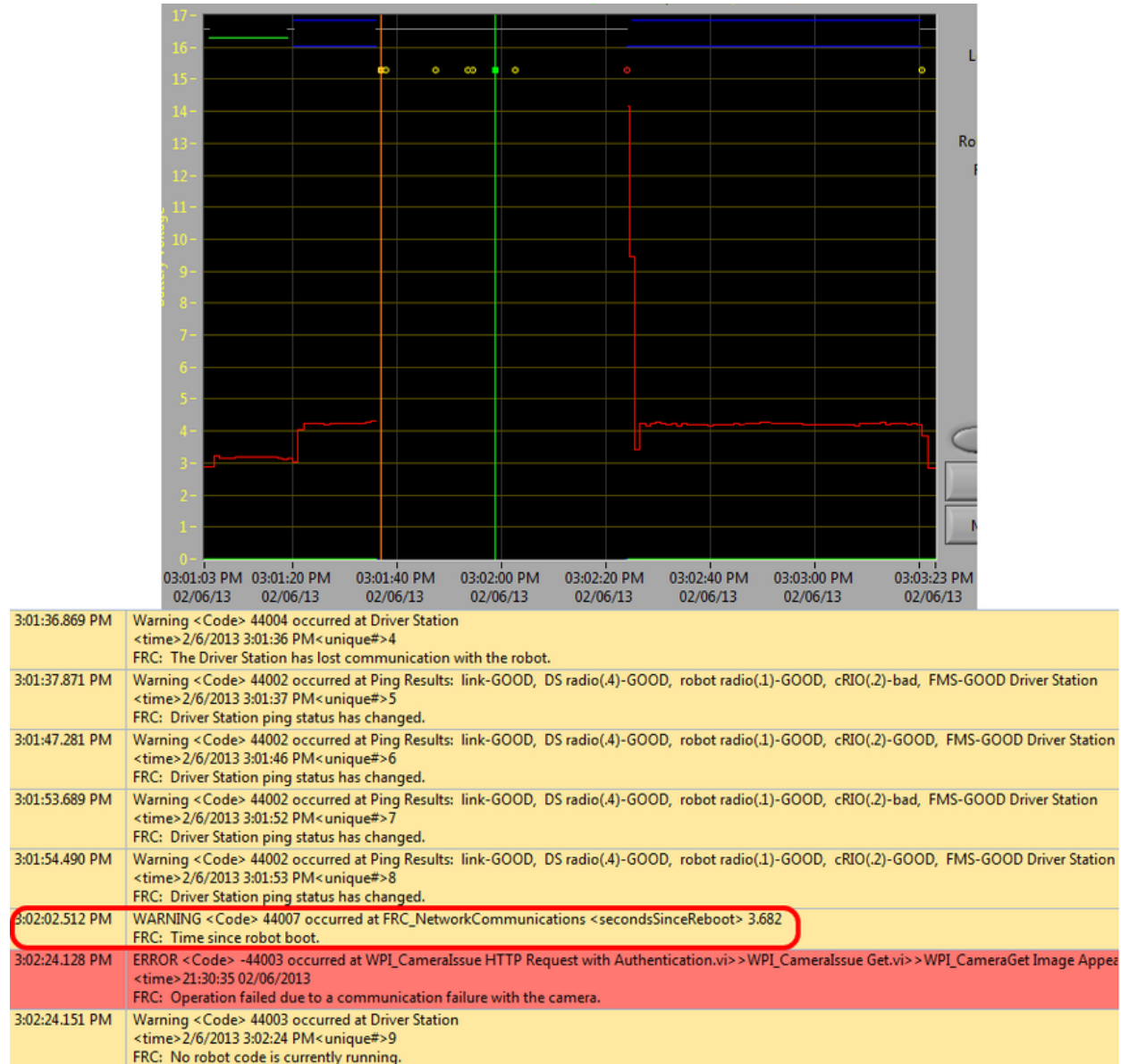
This is an example of a normal match log. The errors and warnings contained in the first box are from when the DS first started and can be ignored. This is confirmed by observing that these events occurred prior to the “FMS Connected:” event. The last event shown can also be ignored, it is also from the robot first connecting to the DS (it occurs 3 seconds after connecting to FMS) and occurs roughly 30 seconds before the match started.

Disconnected from FMS



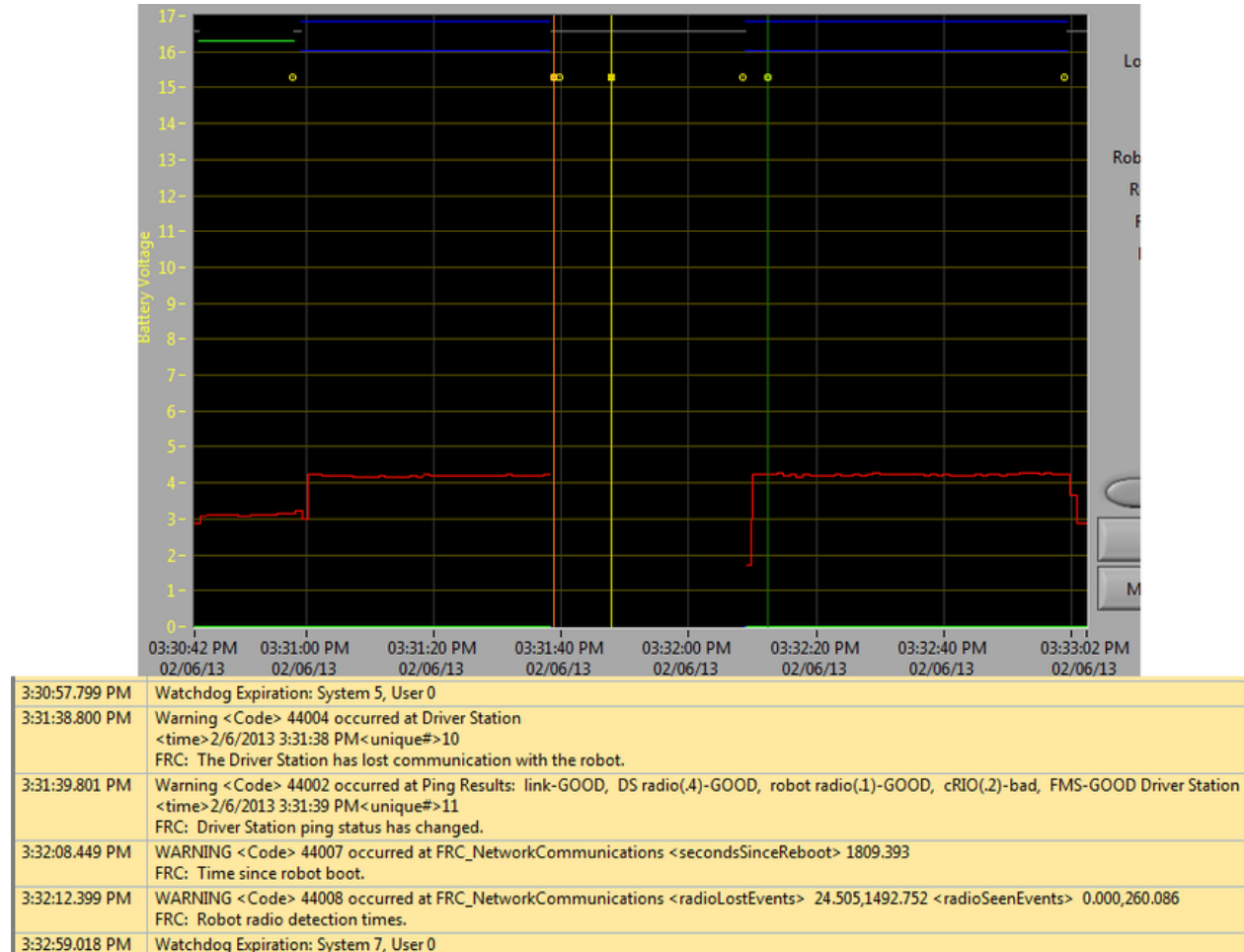
When the DS disconnects from FMS, and therefore the robot, during the match it may segment the log into pieces. The key indicators to this failure are the last event of the first log, indicating that the connection to FMS is now “bad” and the second event from the 2nd log which is a new FMS connected message followed by the DS immediately transitioning into Teleop Enabled. The most common cause of this type of failure is an ethernet cable with no latching tab or a damaged ethernet port on the DS computer.

roboRIO Reboot



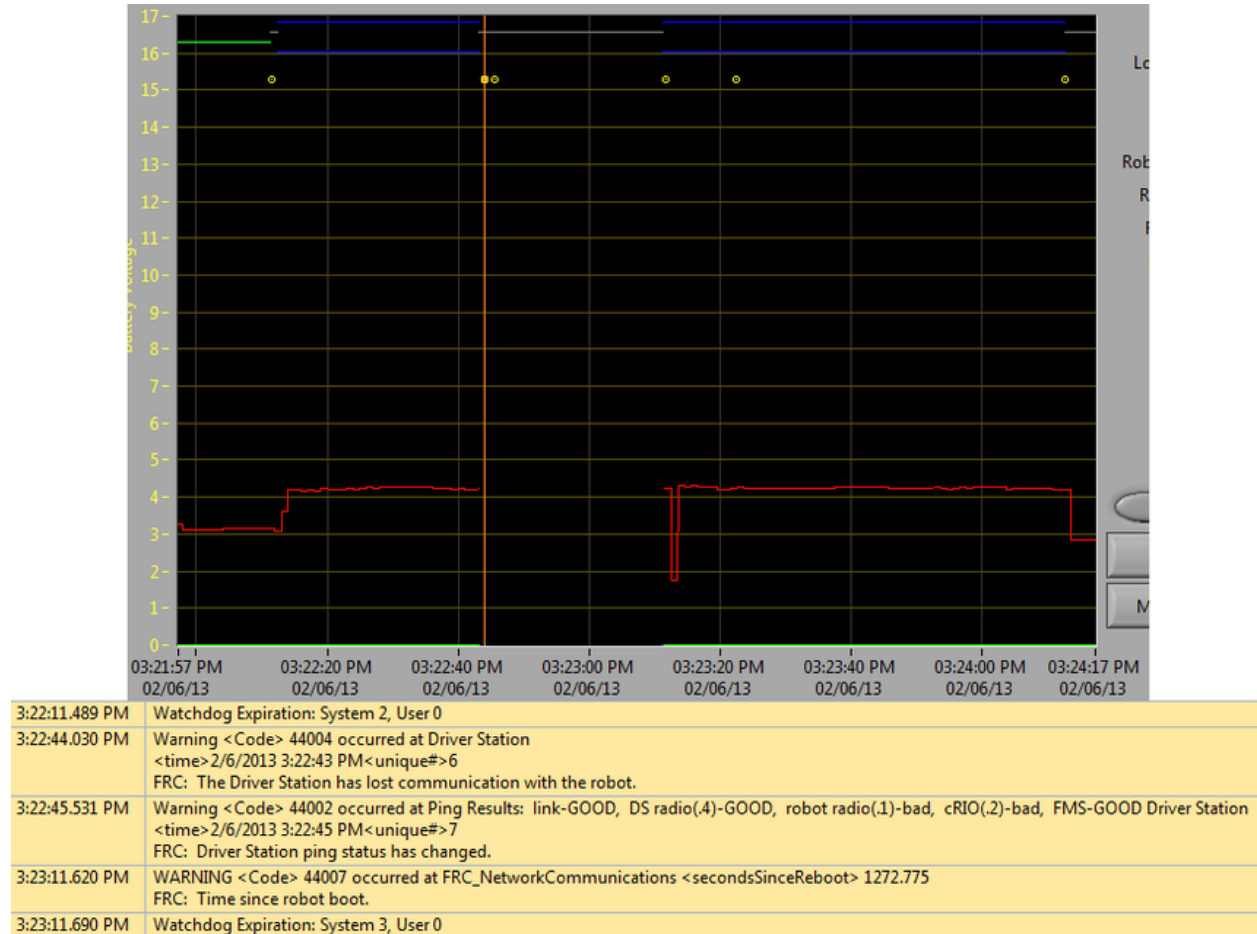
The “Time since robot boot” message is the primary indicator in a connection failure caused by the roboRIO rebooting. In this log the DS loses connection with the roboRIO at 3:01:36 as indicated by the first event. The second event indicates that the ping initiated after the connection failed was successful to all devices other than the roboRIO. At 3:01:47 the roboRIO begins responding to pings again, one additional ping fails at 3:01:52. At 3:02:02 the Driver Station connects to the roboRIO and the roboRIO reports that it has been up for 3.682 seconds. This is a clear indicator that the roboRIO has rebooted. The code continues to load and at 3:02:24 the code reports an error communicating with the camera. A warning is also reported indicating that no robot code is running right before the code finishes starting up.

Ethernet cable issue on robot



An issue with the ethernet cable on the robot is primarily indicated by the ping to the roboRIO going to bad and Radio Lost and Radio Seen events when the roboRIO reconnects. The “Time since robot boot” message when the roboRIO reconnects will also indicate that the roboRIO has not rebooted. In this example, the robot Ethernet cable was disconnected at 3:31:38. The ping status indicates that the D-Link radio is still connected. When the robot reconnects at 3:32:08 the “Tim since robot boot” is 1809 seconds indicating that the roboRIO clearly did not reboot. At 3:32:12 the robot indicates that it lost the radio 24.505 seconds ago and it returned 0.000 seconds ago. These points are plotted as vertical lines on the graph, yellow for radio lost and green for radio seen. Note that the times are slightly offset from the actual events as shown via the disconnection and connection, but help to provide additional information about what is occurring.

Radio reboot



A reboot of the robot radio is typically characterized by a loss of connection to the radio for ~40-45 seconds. In this example, the radio briefly lost power at 3:22:44, causing it to start rebooting. The event at 3:22:45 indicates that the ping to the radio failed. At 3:23:11, the DS regains communication with the roboRIO and the roboRIO indicates it has been up for 1272.775 seconds, ruling out a roboRIO reboot. Note that the network switch on the radio comes back up very quickly so a momentary power loss may not result in a “radio lost”/“radio seen” event pair. A longer disturbance may result in radio events being logged by the DS. In that case, the distinguishing factor which points towards a radio reboot is the ping status of the radio from the DS. If the radio resets, the radio will be unreachable. If the issue is a cabling or connection issue on the robot, the radio ping should remain “GOOD”.

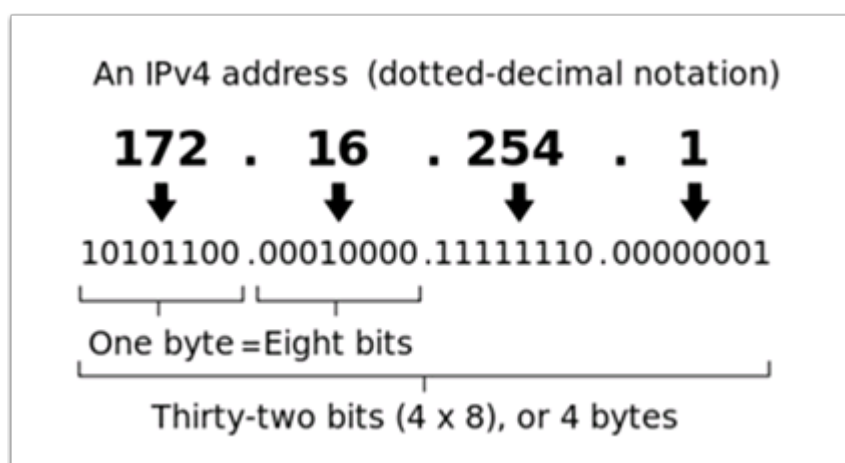
Networking Introduction

This section outlines basic robot configuration and usage relating to communication between the driver station and roboRIO.

21.1 Networking Basics

21.1.1 What is an IP Address?

An IP address is a unique string of numbers, separated by periods that identifies each device on a network. Each IP address is divided up into 4 sections (octets) ranging from 0-255.



As shown above, this means that each IP address is a 32-bit address meaning there are 2^{32} addresses, or nearly 4,300,000,000 addresses possible. However, most of these are used publicly for things like web servers.

This brings up our **first key point** of IP Addressing: Each device on the network must have a unique IP address. No two devices can have the same IP address, otherwise collisions will occur.

Since there are only 4-billion addresses, and there are more than 4-billion computers connected to the internet, we need to be as efficient as possible with giving out IP addresses.

This brings us to public vs. private addresses.

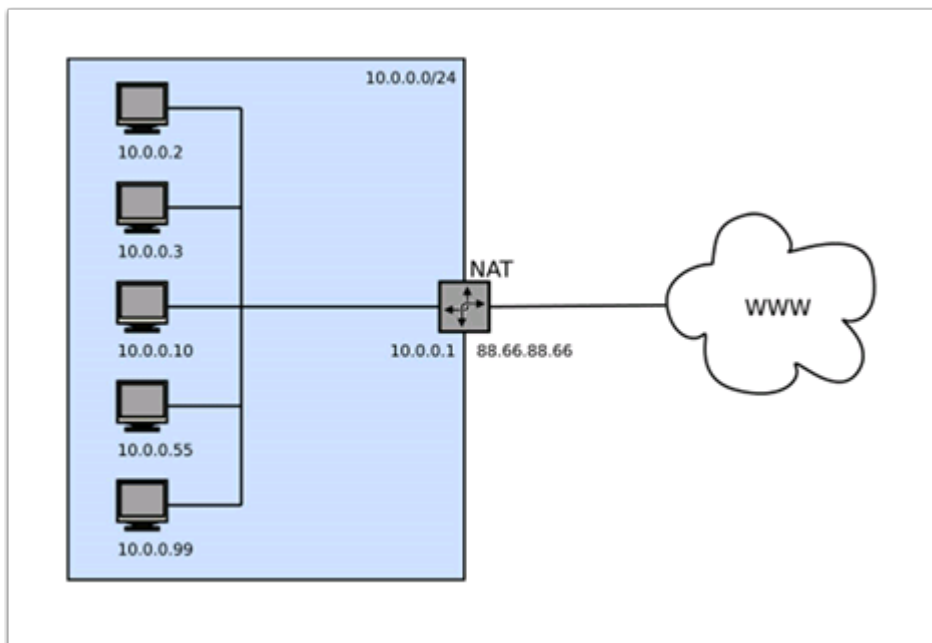
21.1.2 Public vs Private IP Addresses

To be efficient with using IP Addresses, the idea of “Reserved IP Ranges” was implemented. In short, this means that there are ranges of IP Addresses that will never be assigned to web servers, and will only be used for local networks, such as those in your house.

Key point #2: Unless you are directly connecting to your internet provider’s basic modem (no router function), your device will have an IP Address in one of these ranges. This means that at any local network, such as: your school, work office, home, etc., your device will 99% of the time have an IP address in a range listed below:

Class	Bits	Start Address	End Address	Number of Addresses
A	24	10.0.0.0	10.255.255.255	16,777,216
B	20	172.16.0.0	172.31.255.255	1,048,576
C	16	192.168.0.0	192.168.255.255	65,536

These reserved ranges let us assign one “unreserved IP Address” to an entire house, and then use multiple addresses in a reserved range to connect more than one computer to the internet. A process on the home’s internet router known as **NAT** (Network Address Translation), handles the process of keeping track which private IP is requesting data, using the public IP to request that data from the internet, and then passing the returned data back to the private IP that requested it. This allows us to use the same reserved IP addresses for many local networks, without causing any conflicts. An image of this process is presented below.



Note: For the FRC networks, we will use the 10.0.0.0 range. This range allows us to use the 10.TE.AM.xx format for IP addresses, whereas using the Class B or C networks would only allow a subset of teams to follow the format. An example of this formatting would be 10.17.50.1 for FRC Team 1750.

21.1.3 How are these addresses assigned?

We've covered the basics of what IP addresses are, and which IP addresses we will use for the FRC competition, so now we need to discuss how these addresses will get assigned to the devices on our network. We already stated above that we can't have two devices on the same network with the same IP Address, so we need a way to be sure that every device receives an address without overlapping. This can be done Dynamically (automatic), or Statically (manual).

Dynamically

Dynamically assigning IP addresses means that we are letting a device on the network manage the IP address assignments. This is done through the Dynamic Host Configuration Protocol (DHCP). DHCP has many components to it, but for the scope of this document, we will think of it as a service that automatically manages the network. Whenever you plug in a new device to the network, the DHCP service sees the new device, then provides it with an available IP address and the other network settings required for the device to communicate. This can mean that there are times we do not know the exact IP address of each device.

What is a DHCP server?

A DHCP server is a device that runs the DHCP service to monitor the network for new devices to configure. In larger businesses, this could be a dedicated computer running the DHCP service and that computer would be the DHCP server. For home networks, FRC networks, and other smaller networks, the DHCP service is usually running on the router; in this case, the router is the DHCP server.

This means that if you ever run into a situation where you need to have a DHCP server assigning IP addresses to your network devices, it's as simple as finding the closest home router, and plugging it in.

Statically

Statically assigning IP addresses means that we are manually telling each device on the network which IP address we want it to have. This configuration happens through a setting on each device. By disabling DHCP on the network and assigning the addresses manually, we get the benefit of knowing the exact IP address of each device on the network, but because we set each one manually and there is no service keeping track of the used IP addresses, we have to keep track of this ourselves. While statically setting IP addresses, we must be careful not to assign duplicate addresses, and must be sure we are setting the other network settings (such as subnet mask and default gateway) correctly on each device.

21.1.4 What is link-local?

If a device does not have an IP address, then it cannot communicate on a network. This can become an issue if we have a device that is set to dynamically acquire its address from a DHCP server, but there is no DHCP server on the network. An example of this would be when you have a laptop directly connected to a roboRIO and both are set to dynamically acquire an IP address. Neither device is a DHCP server, and since they are the only two devices on the network, they will not be assigned IP addresses automatically.

Link-local addresses give us a standard set of addresses that we can “fall-back” to if a device set to acquire dynamically is not able to acquire an address. If this happens, the device will assign itself an IP address in the 169.254.xx.yy address range; this is a link-local address. In our roboRIO and computer example above, both devices will realize they haven’t been assigned an IP address and assign themselves a link-local address. Once they are both assigned addresses in the 169.254.xx.yy range, they will be in the same network and will be able to communicate, even though they were set to dynamic and a DHCP server did not assign addresses.

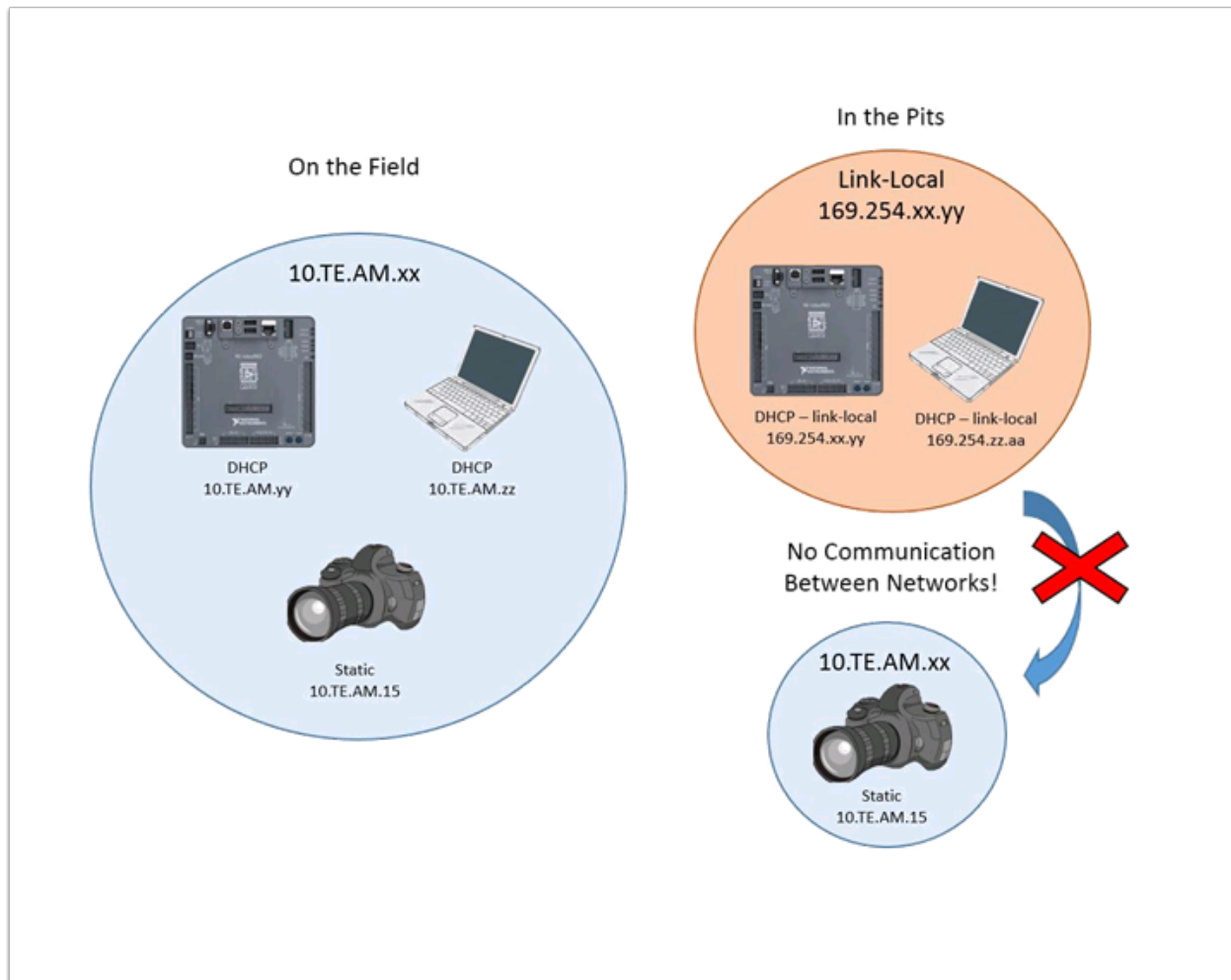
21.1.5 IP Addressing for FRC

See the [*IP Networking Article*](#) for more information.

Mixing Dynamic and Static Configurations

While on the field, the team should not notice any issues with having devices set statically in the 10.TE.AM.xx range, and having the field assign DHCP addresses as long as there are no IP address conflicts as referred to in the section above.

In the pits, a team may encounter issues with mixing Static and DHCP devices for the following reason. As mentioned above, DHCP devices will fall back to a link-local address (169.254.xx.yy) if a server isn’t present. For static devices, the IP address will always be the same. If the DHCP server is not present and the roboRIO, driver station, and laptop fall back to link-local addresses, the statically set devices in the 10.TE.AM.xx range will be in a different network and not visible to those with link-local addresses. A visual description of this is provided below:



21.1.6 mDNS

mDNS, or multicast Domain Name System is a protocol that allows us to benefit from the features of DNS, without having a DNS server on the network. To make this clearer, let's take a step back and talk about what DNS is.

What is DNS?

DNS (Domain Name System) can become a complex topic, but for the scope of this paper, we are going to just look at the high level overview of DNS. In the most basic explanation, DNS is what allows us to relate human-friendly names for network devices to IP Addresses, and keep track of those IP addresses if they change.

Example 1: Let's look at the site www.google.com. The IP address for this site is 172.217.164.132, however that is not very user friendly to remember!

Whenever a user types www.google.com into their computer, the computer contacts the DNS server (a setting provided by DHCP!) and asks what is the IP address on file for www.google.com. The DNS server returns the IP address and then the computer is able to use that to connect to the Google web site.

Example 2: On your home network, you have a server named MYCOMPUTER that you want to connect to from your laptop. Your network uses DHCP so you don't know the IP Address of MYCOMPUTER, but DNS allows you to connect just by using the MYCOMPUTER name. Additionally, whenever the DHCP assignments refresh, MYCOMPUTER may end up with a different address, but because you're connecting by using the MYCOMPUTER name instead of a specific IP address, the DNS record was updated and you're still able to connect.

This is the second benefit to DNS, and the most relevant for FRC. With DNS, if we reference devices by their friendly name instead of IP Address, we don't have to change anything in our program if the IP Address changes. DNS will keep track of the changes and return the new address if it ever changes.

DNS for FRC

On the field and in the pits, there is no DNS server that allows us to perform the lookups like we do for the Google web site, but we'd still like to have the benefits of not remembering every IP Address, and not having to guess at every device's address if DHCP assigns a different address than we expect. This is where mDNS comes into the picture.

mDNS provides us the same benefits as traditional DNS, but is just implemented in a way that does not require a server. Whenever a user asks to connect to a device using a friendly name, mDNS sends out a message asking the device with that name to identify itself. The device with the name then sends a return message including its IP address so all devices on the network can update their information. mDNS is what allows us to refer to our roboRIO as `roboRIO-TEAM-FRC.local` and have it connect on a DHCP network.

Note: If a device used for FRC does not support mDNS, then it will be assigned an IP Address in the 10.TE.AM.20 - 10.TE.AM.255 range, but we won't know the exact address to connect and we won't be able to use the friendly name like before. In this case, the device would need to have a static IP Address.

mDNS - Principles

Multicast Domain Name System (mDNS) is a system which allows for resolution of host names to IP addresses on small networks with no dedicated name server. To resolve a host name a device sends out a multicast message to the network querying for the device. The device then responds with a multicast message containing its IP. Devices on the network can store this information in a cache so subsequent requests for this address can be resolved from the cache without repeating the network query.

mDNS - Providers

To use mDNS, an mDNS implementation is required to be installed on your PC. Here are some common mDNS implementations for each major platform:

Windows:

- **NI mDNS Responder:** Installed with the NI FRC Game Tools
- **Apple Bonjour:** Installed with iTunes

OSX:

- **Apple Bonjour:** Installed by default

Linux:

- **nss-mDNS/Avahi/Zeroconf:** Installed and enabled by default on some Linux variants (such as Ubuntu or Mint). May need to be installed or enabled on others (such as Arch)

mDNS - Firewalls

Note: Depending on your PC configuration, no changes may be required, this section is provided to assist with troubleshooting.

To work properly mDNS must be allowed to pass through your firewall. Because the network traffic comes from the mDNS implementation and not directly from the Driver Station or IDE, allowing those applications through may not be sufficient. There are two main ways to resolve mDNS firewall issues:

- Add an application/service exception for the mDNS implementation (NI mDNS Responder is C:\Program Files\National Instruments\Shared\mDNS Responder\nimdnsResponder.exe)
- Add a port exception for traffic to/from UDP 5353. IP Ranges:
 - 10.0.0.0 - 10.255.255.255
 - 172.16.0.0 - 172.31.255.255
 - 192.168.0.0 - 192.168.255.255
 - 169.254.0.0 - 169.254.255.255
 - 224.0.0.251

mDNS - Browser support

Most web-browsers should be able to utilize the mDNS address to access the roboRIO web server as long as an mDNS provider is installed. These browsers include Microsoft Edge, Firefox, and Google Chrome.

21.1.7 USB

If using the USB interface, no network setup is required (you do need the *Installing the FRC Game Tools* installed to provide the roboRIO USB Driver). The roboRIO driver will automatically configure the IP address of the host (your computer) and roboRIO and the software listed above should be able to locate and utilize your roboRIO.

21.1.8 Ethernet/Wireless

The *Programming your Radio* will enable the DHCP server on the OpenMesh radio in the home use case (AP mode), if you are putting the OpenMesh in bridge mode and using a router, you can enable DHCP addressing on the router. The bridge is set to the same team based IP

address as before (10.TE.AM.1) and will hand out DHCP address from 10.TE.AM.20 to 10.TE.AM.199. When connected to the field, FMS will also hand out addresses in the same IP range.

21.1.9 Summary

IP Addresses are what allow us to communicate with devices on a network. For FRC, these addresses are going to be in the 10.TE.AM.xx range if we are connected to a DHCP server or if they are assigned statically, or in the link-local 169.254.xx.yy range if the devices are set to DHCP, but there is no server present. For more information on how IP Addresses work, see [this](#) article by Microsoft.

If all devices on the network support mDNS, then all devices can be set to DHCP and referred to using their friendly names (ex. roboRIO-TEAM-FRC.local). If some devices do not support mDNS, they will need to be set to use static addresses.

If all devices are set to use DHCP or Static IP assignments (with correct static settings), the communication should work in both the pit and on the field without any changes needed. If there are a mix of some Static and some DHCP devices, then the Static devices will connect on the field, but will not connect in the pit. This can be resolved by either setting all devices to static settings, or leaving the current settings and providing a DHCP server in the pit.

21.2 IP Configurations

Note: This document describes the IP configuration used at events, both on the fields and in the pits, potential issues and workaround configurations.

21.2.1 TE.AM IP Notation

The notation TE.AM is used as part of IPs in numerous places in this document. This notation refers to splitting your four digit team number into two digit pairs for the IP address octets.

Example: 10.TE.AM.2

Team 12 - 10.0.12.2

Team 122 - 10.1.22.2

Team 1212 - 10.12.12.2

Team 3456 - 10.34.56.2

21.2.2 On the Field

This section describes networking when connected to the Field Network for match play

On the Field DHCP Configuration

The Field Network runs a DHCP server with pools for each team that will hand out addresses in the range of 10.TE.AM.20 and up with subnet masks

- OpenMesh OM5P-AN or OM5P-AC radio - Static 10.TE.AM.1 programmed by Kiosk
- roboRIO - DHCP 10.TE.AM.2 assigned by the Robot Radio
- Driver Station - DHCP (“Obtain an IP address automatically”) 10.TE.AM.X assigned by field
- IP camera (if used) - DHCP 10.TE.AM.Y assigned by Robot Radio
- Other devices (if used) - DHCP 10.TE.AM.Z assigned by Robot Radio

On the Field Static Configuration

It is also possible to configure static IPs on your devices to accommodate devices or software which do not support mDNS. When doing so you want to make sure to avoid addresses that will be in use when the robot is on the field network. These addresses are 10.TE.AM.1 and 10.TE.AM.4 for the OpenMesh radio and the field access point and anything 10.TE.AM.20 and up which may be assigned to a device still configured for DHCP. The roboRIO network configuration can be set from the webdashboard.

- OpenMesh radio - Static 10.TE.AM.1 programmed by Kiosk
- roboRIO - Static 10.TE.AM.2 would be a reasonable choice, subnet mask of 255.255.255.0 (default)
- Driver Station - Static 10.TE.AM.5 would be a reasonable choice, subnet mask **must** be 255.0.0.0
- IP Camera (if used) - Static 10.TE.AM.11 would be a reasonable choice, subnet 255.255.255.0 should be fine
- Other devices - Static 10.TE.AM.6-.10 or .12-.19 (.11 if camera not present) subnet 255.255.255.0

21.2.3 In the Pits

Note: New for 2018: There is now a DHCP server running on the wired side of the Robot Radio in the event configuration.

In the Pits DHCP Configuration

- OpenMesh radio - Static 10.TE.AM.1 programmed by Kiosk.
- roboRIO - 10.TE.AM.2, assigned by Robot Radio
- Driver Station - DHCP (“Obtain an IP address automatically”), 10.TE.AM.X, assigned by Robot Radio
- IP camera (if used) - DHCP, 10.TE.AM.Y, assigned by Robot Radio

- Other devices (if used) - DHCP, 10.TE.AM.Z, assigned by Robot Radio

In the Pits Static Configuration

It is also possible to configure static IPs on your devices to accommodate devices or software which do not support mDNS. When doing so you want to make sure to avoid addresses that will be in use when the robot is on the field network. These addresses are 10.TE.AM.1 and 10.TE.AM.4 for the OpenMesh radio

21.3 roboRIO Network Troubleshooting

The roboRIO and the 2015 FRC tools use dynamic IP addresses (DHCP) for network connectivity. This article describes steps for troubleshooting networking connectivity between your PC and your roboRIO

21.3.1 Ping roboRIO

The first step to identifying roboRIO networking issues is to isolate if it is an application issue or a general network issue. To do this, click **Start -> type cmd -> press Enter** to open the command prompt. Type `ping roboRIO-####-FRC.local` where #### is your team number (with no leading zeroes) and press enter. If the ping succeeds, the issue is likely with the specific application, verify your team number configuration in the application, and check your firewall configuration.

21.3.2 USB Connection Troubleshooting

If you are attempting to troubleshoot the USB connection, try pinging the roboRIO's IP address. As long as there is only one roboRIO connected to the PC, it should be configured as 172.22.11.2. If this ping fails, make sure you have the roboRIO connected and powered, and that you have installed the NI FRC Game Tools. The game tools installs the roboRIO drivers needed for the USB connection.

If this ping succeeds, but the .local ping fails, it is likely that either the roboRIO hostname is configured incorrectly, or you are connected to a DNS server which is attempting to resolve the .local address.

- Verify that your roboRIO has been imaged for your team number. This sets the hostname used by mDNS.
- Disconnect your computer from all other networks including Ethernet and WiFi. It is possible that one of these networks contains a DNS server that is attempting to resolve the .local address.

21.3.3 Ethernet Connection

The screenshot displays the 'roboRIO-40 : System Configuration' web interface. On the left is a sidebar with navigation icons. The main area is divided into three sections: 'System Settings', 'Startup Settings', and 'System Resources'. In the 'System Settings' section, the 'IP Address' field is highlighted with a red circle, showing '10.0.40.2 (Ethernet)'. Below it, '172.22.11.2 (Ethernet)' is also listed. Other fields include 'Hostname' (roboRIO-40), 'DNS Name', 'Vendor' (National Instruments), 'Model' (roboRIO), 'Serial Number' (030498A9), 'Firmware Revision' (2.0.0f1), 'Operating System' (NI Linux Real-Time ARMv7-A 3.2.35-rt52-2.0.0f0), 'Status' (Running), 'System Start Time' (10/1/2014 2:15:56 PM), 'Image Title' (roboRIO Image), 'Image Version' (FRC_roboRIO_2015_v14), 'Comments', and 'Locale' (English). The 'Startup Settings' section contains several checkboxes: 'Force Safe Mode' (unchecked), 'Enable Console Out' (checked), 'Disable RT Startup App' (unchecked), 'Disable FPGA Startup App' (unchecked), 'Enable Secure Shell Server (sshd)' (checked), and 'LabVIEW Project Access' (checked). The 'System Resources' section shows 'Total Physical Memory' (232 MB), 'Free Physical Memory' (103 MB), and 'Total Virtual Memory' (232 MB). At the bottom right of the 'System Settings' section is an 'Update Firmware' button.

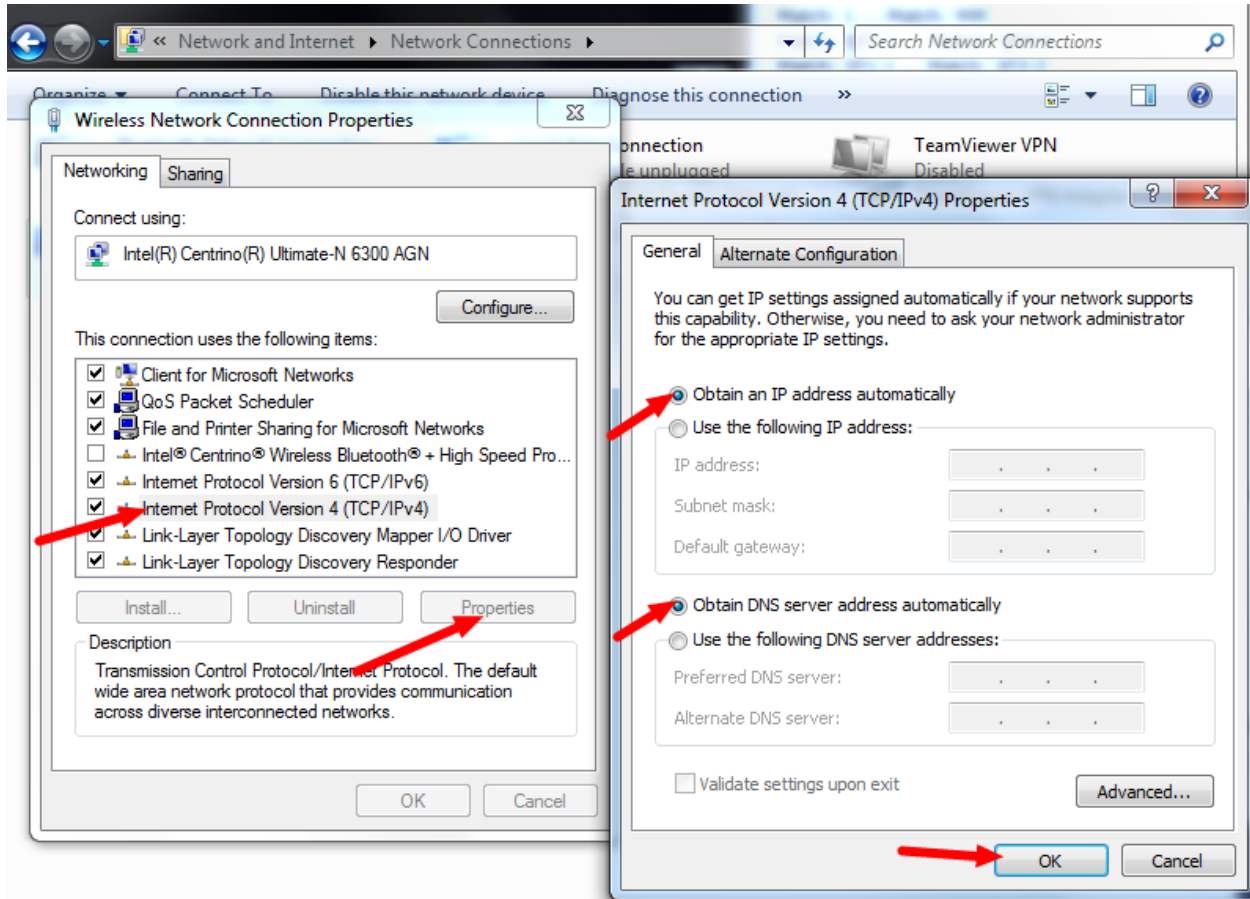
If you are troubleshooting an Ethernet connection, it may be helpful to first make sure that you can connect to the roboRIO using the USB connection. Using the USB connection, open the roboRIO webdashboard and verify that the roboRIO has an IP address on the ethernet interface. If you are tethering to the roboRIO directly this should be a self-assigned 169.*.*.* address, if you are connected to the OM5P-AN radio, it should be an address of the form 10.TE.AM.XX where TEAM is your four digit FRC team number. If the only IP address here is the USB address, verify the physical roboRIO ethernet connection.

Ping the roboRIO IP address

If there is an IP address in the step above, try pinging this IP address using the command prompt as described above. If this works, you have an issue resolving the mDNS address on your PC. The two most common causes are not having an mDNS resolver installed on the system and a DNS server on the network that is trying to resolve the .local address using regular DNS.

- Verify that you have an mDNS resolver installed on your system. On Windows, this is typically fulfilled by the NI FRC Game Tools. For more information on mDNS resolvers, see the [roboRIO Networking article](#).
- Disconnect your computer from any other networks and make sure you have the OM5P-AN configured as an access point, using the [FRC Radio Configuration Utility](#). Removing any other routers from the system will help verify that there is not a DNS server causing the issue.

Ping fails



If pinging the IP address directly fails, you may have an issue with the network configuration of the PC. The PC should be configured to **Obtain an Address Automatically** (also known as DHCP). To check this, click **Start -> Control Panel -> Network Connections -> Change adapter settings**, then right click on the appropriate interface (usually Local Area Connection for Ethernet or Wireless Network Connection for wireless) and select **Properties**. Click **Internet Protocol Version 4**, then click **Properties**. Make sure both radio buttons are set to **Obtain automatically**.

21.3.4 Other things to check

Other possibilities that may cause issues include:

- Proxies. Having a proxy enabled may cause issues with the roboRIO networking.

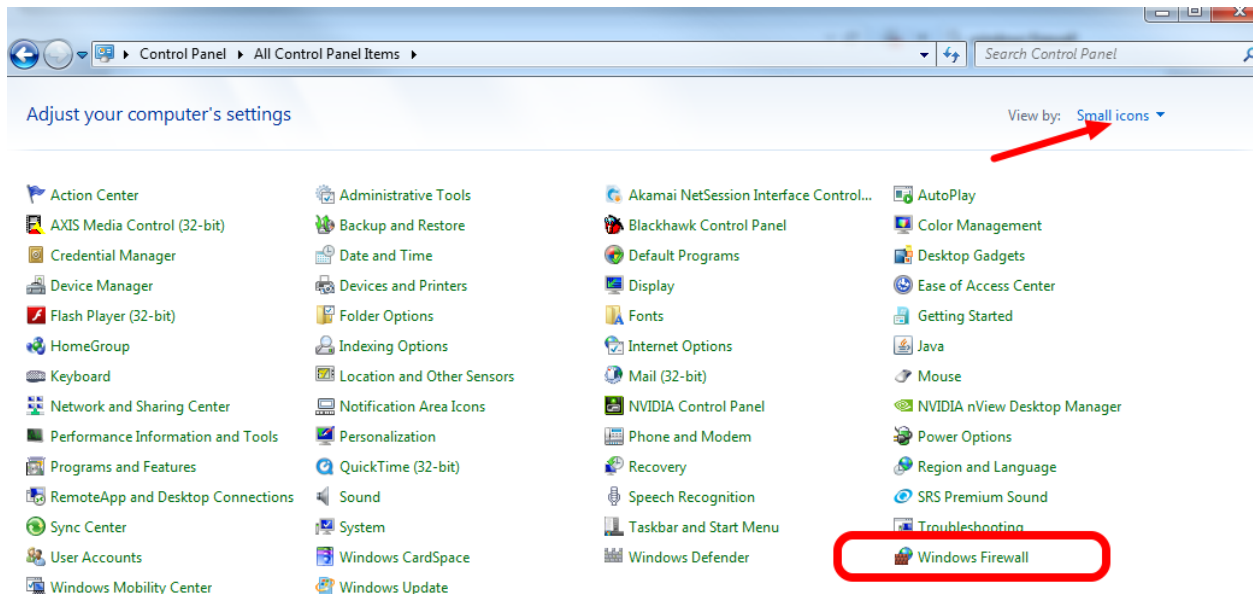
21.4 Windows Firewall Configuration

Many of the programming tools used in FRC need network access for various reasons. Depending on the exact configuration, the Windows Firewall may potentially interfere with this access for one or more of these programs. This document describes procedures for Windows 7, but future versions should be similar.

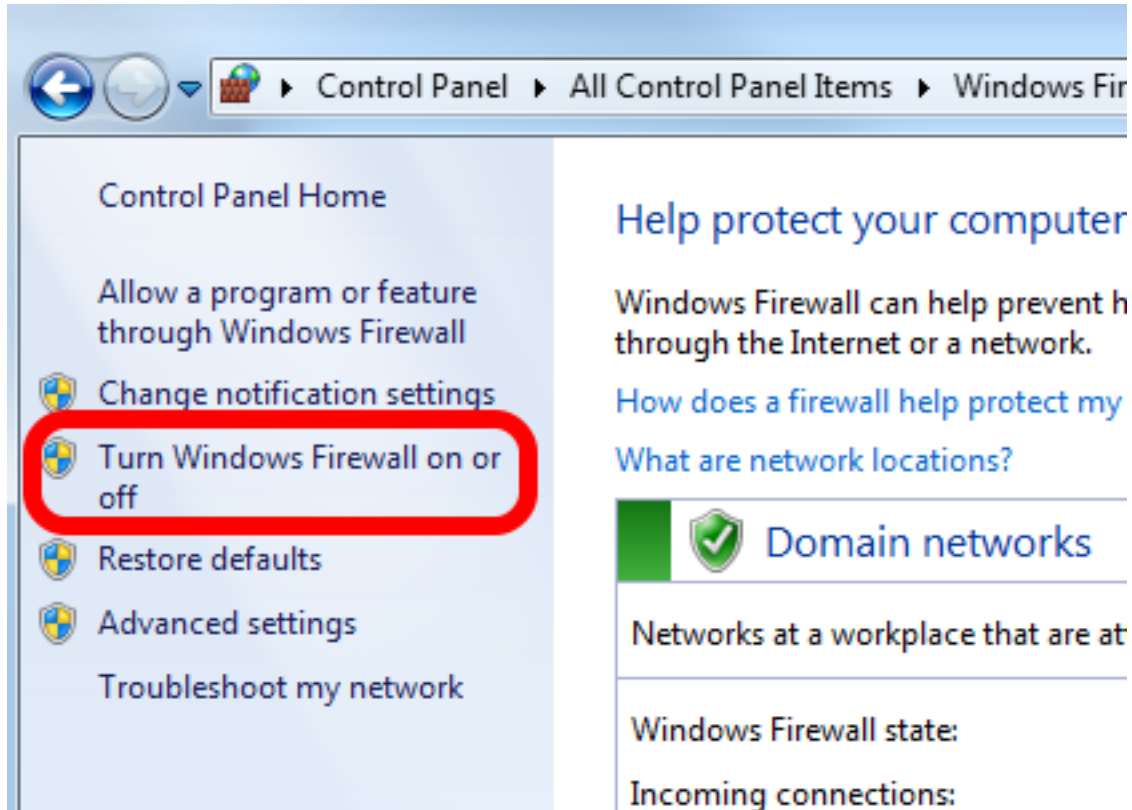
21.4.1 Disabling Windows Firewall

The easiest solution is to disable the Windows Firewall. Teams should beware that this does make the PC potentially more vulnerable to malware attacks if connecting to the internet.

Control Panel

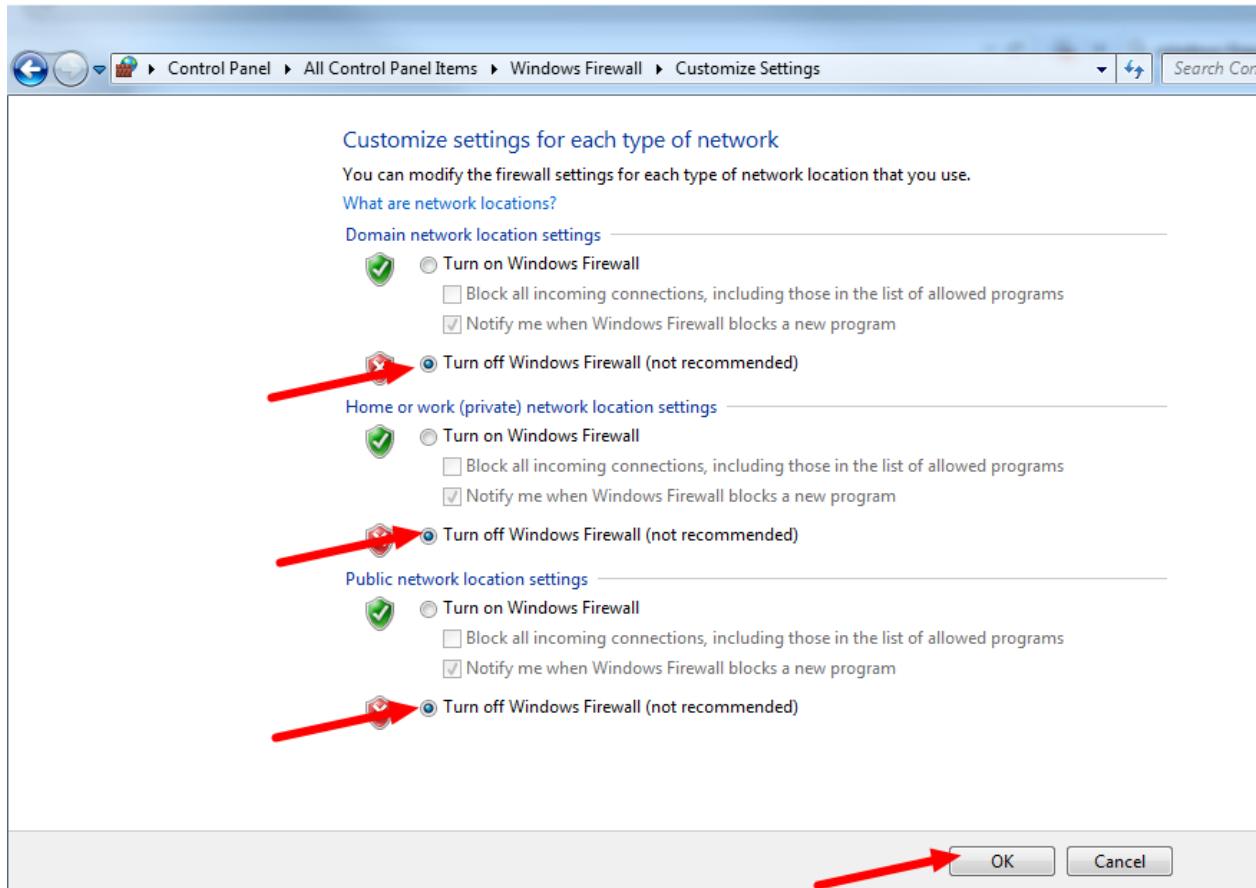


Click Start -> Control Panel to open the Control Panel. Click the dropdown next to View by: and select Small icons then click Windows Defender Firewall.

Turn Windows Firewall on or off

In the left pane, click Turn Windows Defender Firewall on or off, and click yes. Enter your Administrator password if a dialog appears.

Disable the Firewall

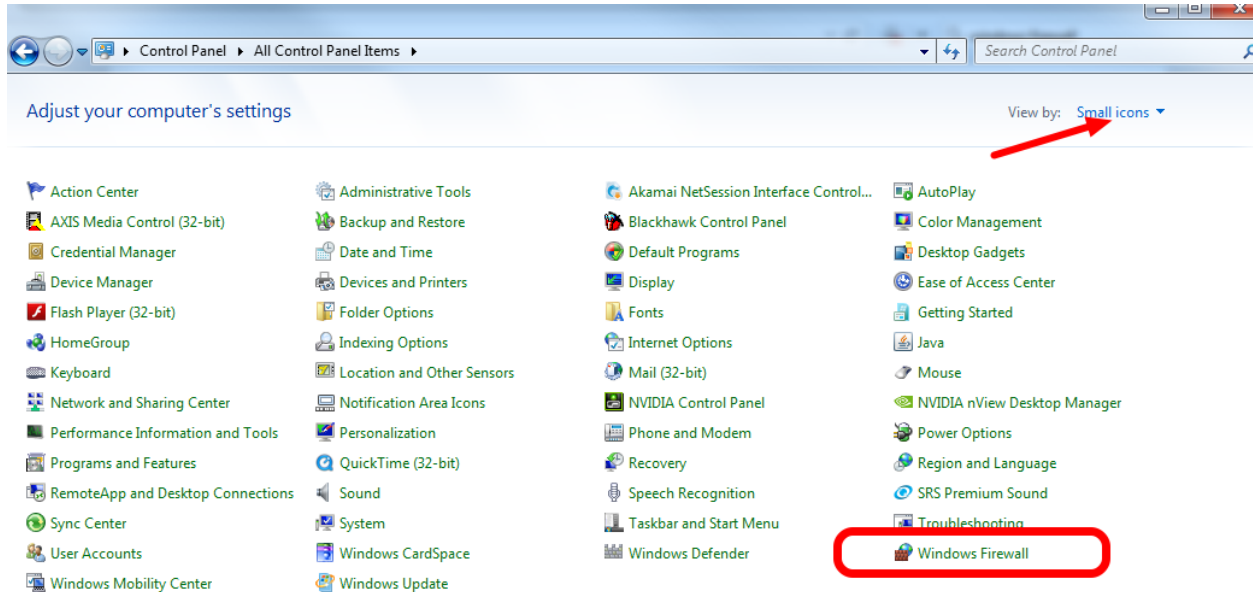


For each category, select the radio button to Turn off Windows Defender Firewall. Then click OK.

21.4.2 Configure the firewall

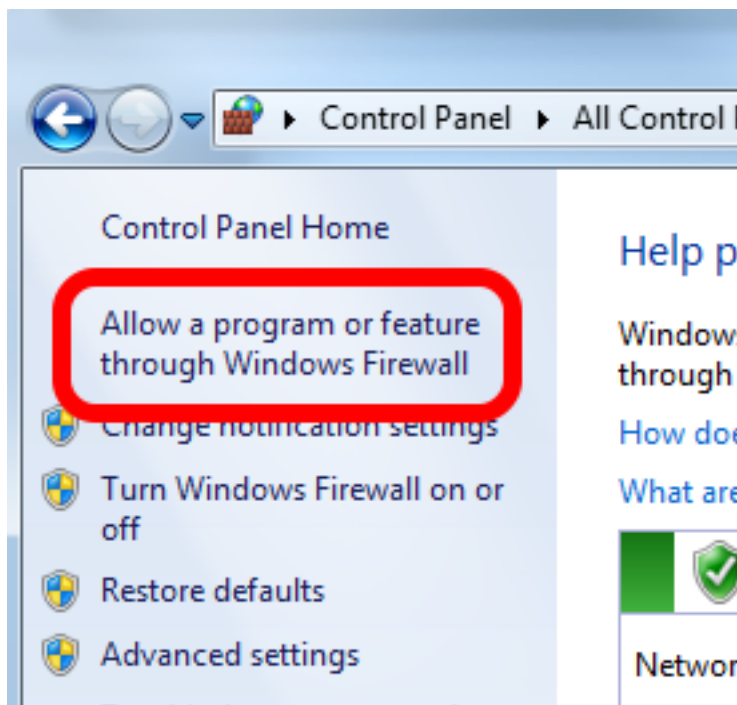
Alternatively, you can add exceptions to the Firewall for any FRC programs you are having issues with.

Open Control Panel



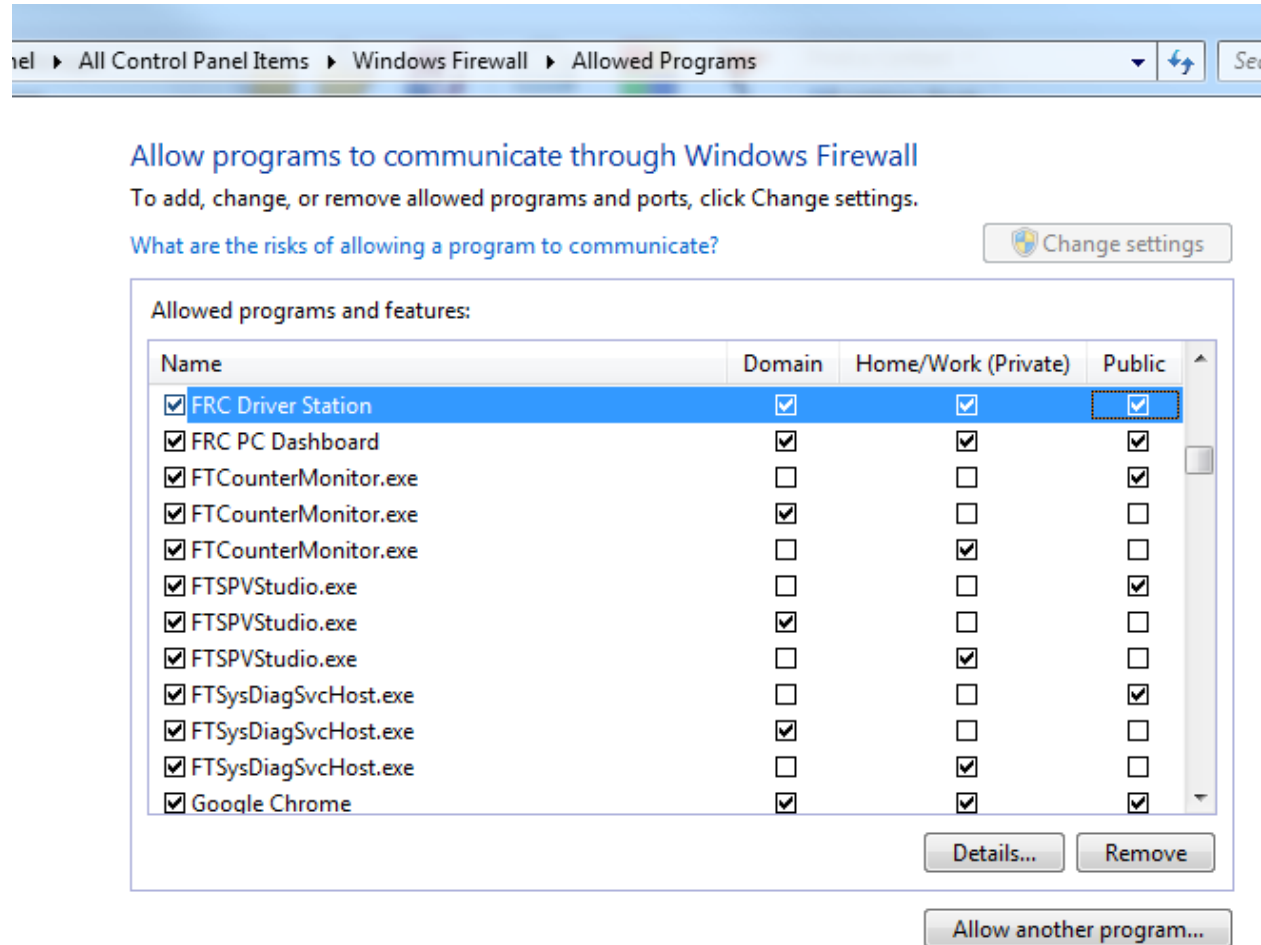
Click Start -> Control Panel to open the Control Panel. Click the dropdown next to View by: and select Small icons then click Windows Defender Firewall.

Allow a program...



In the left pane, click Allow a program or feature through Windows Defender Firewall

Allowed Programs



For each FRC program you are having an issue with, make sure that it appears in the list and that it has a check in each of the 3 columns. If you need to change a setting, you made need to click the [Change settings](#) button in the top right before changing the settings. If the program is not in the list at all, click the [Allow another program...](#) button and browse to the location of the program to add it.

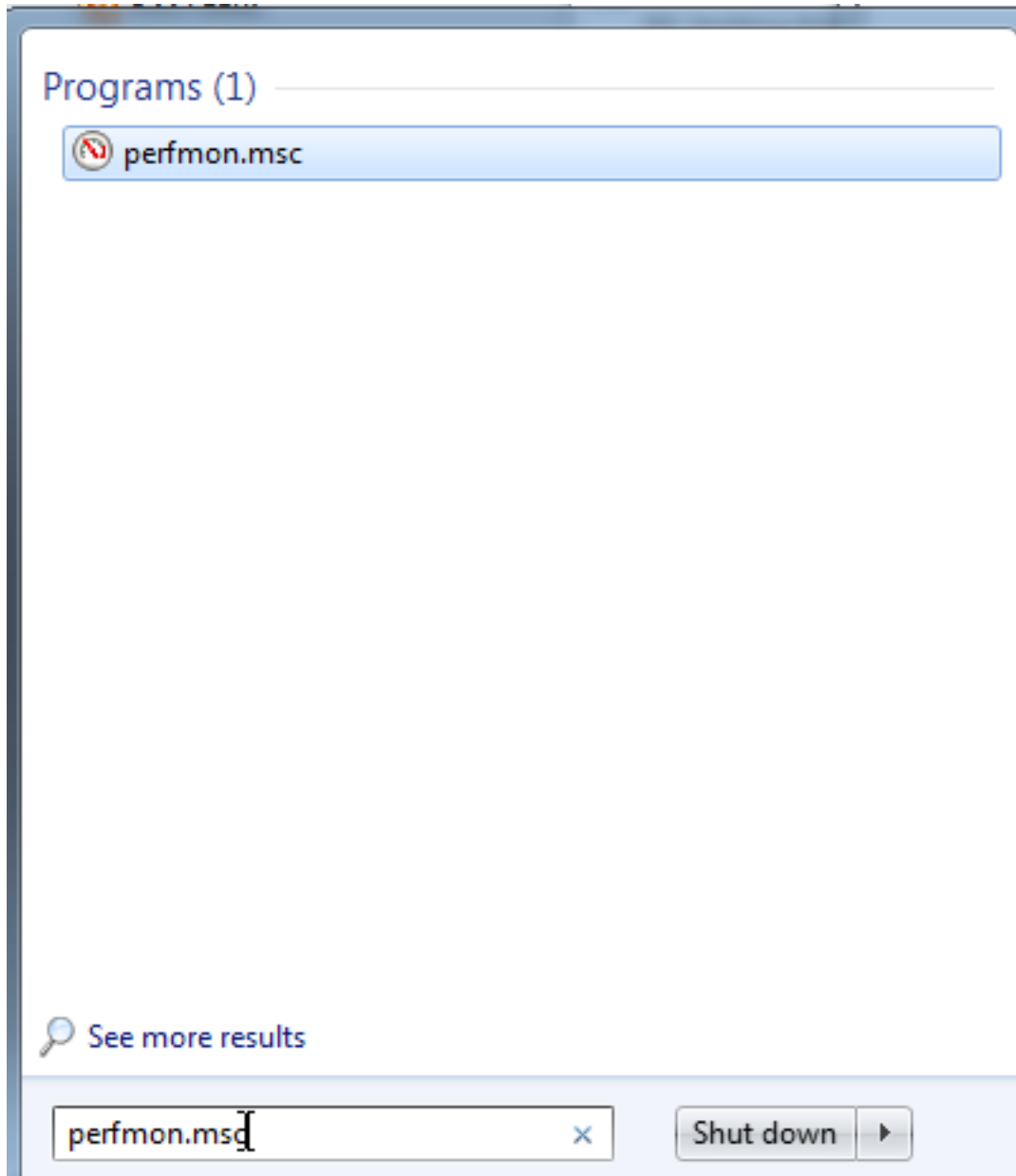
21.5 Measuring Bandwidth Usage

On the FRC Field (and at home when the radio is configured using the FRC Bridge Configuration Utility) each team is limited to 4Mb/s of network traffic (see the [FMS Whitepaper](#) for more details). The FMS Whitepaper provides information on determining the bandwidth usage of the Axis camera, but some teams may wish to measure their overall bandwidth consumption. This document details how to make that measurement.

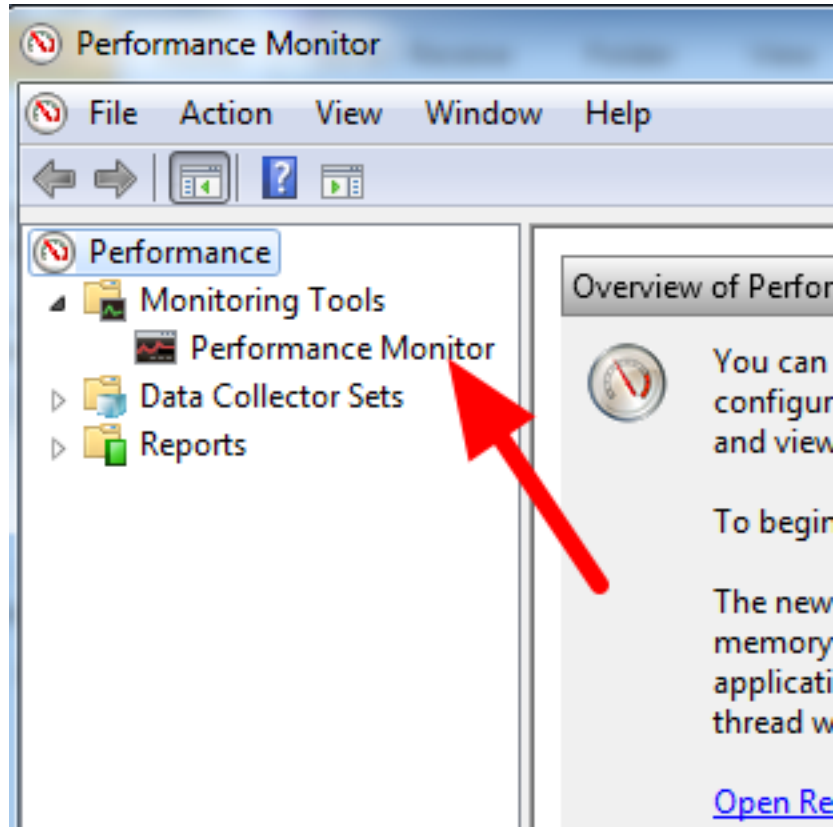
21.5.1 Measuring Bandwidth Using the Performance Monitor (Win 7 only)

Windows 7 contains a built-in tool called the Performance Monitor that can be used to monitor the bandwidth usage over a network interface.

Launching the Performance Monitor

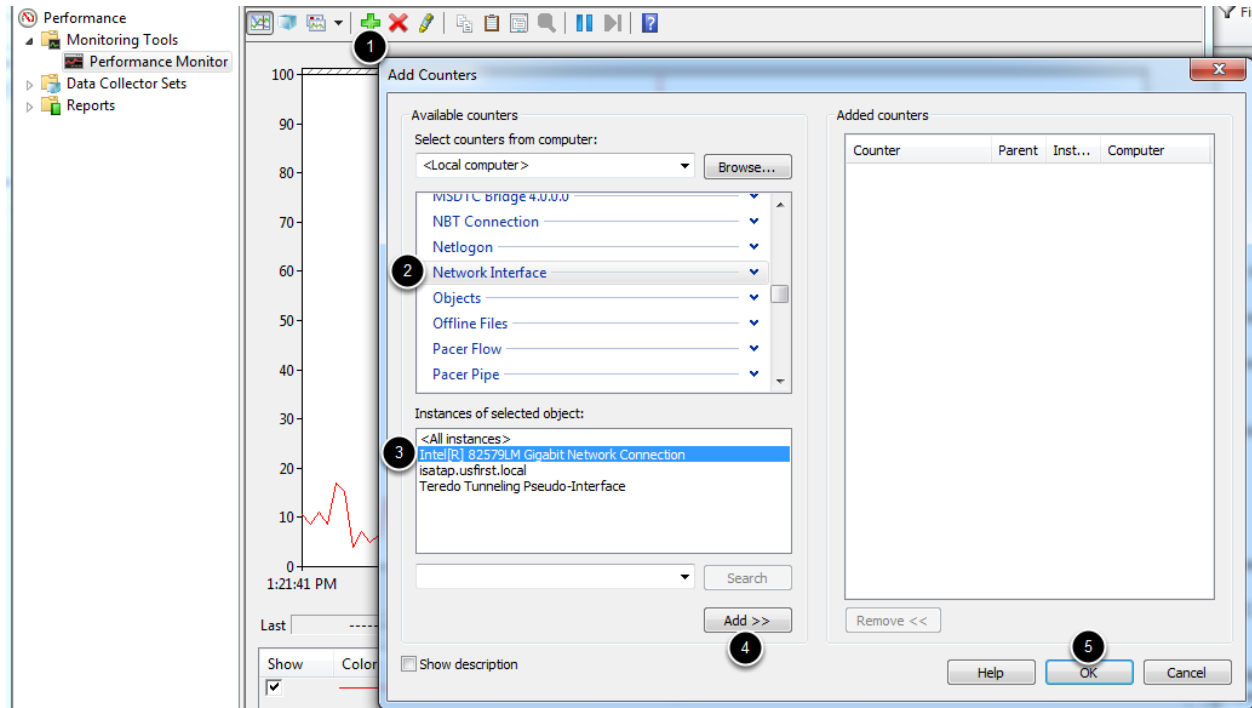


Click Start and in the search box, type `perfmon.msc` and press Enter.

Open Real-Time Monitor

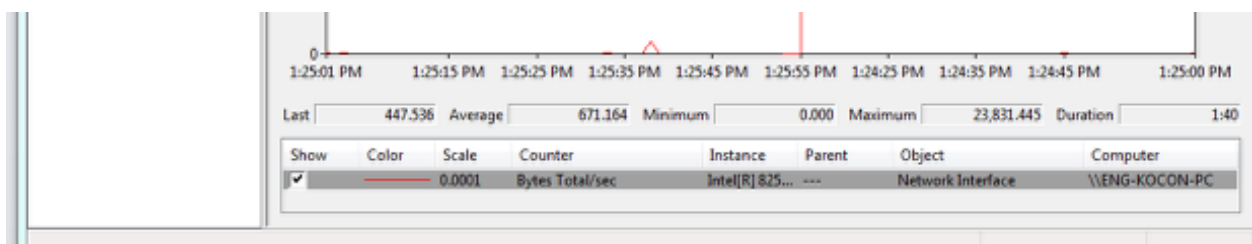
In the left pane, click Performance Monitor to display the real-time monitor.

Add Network Counter



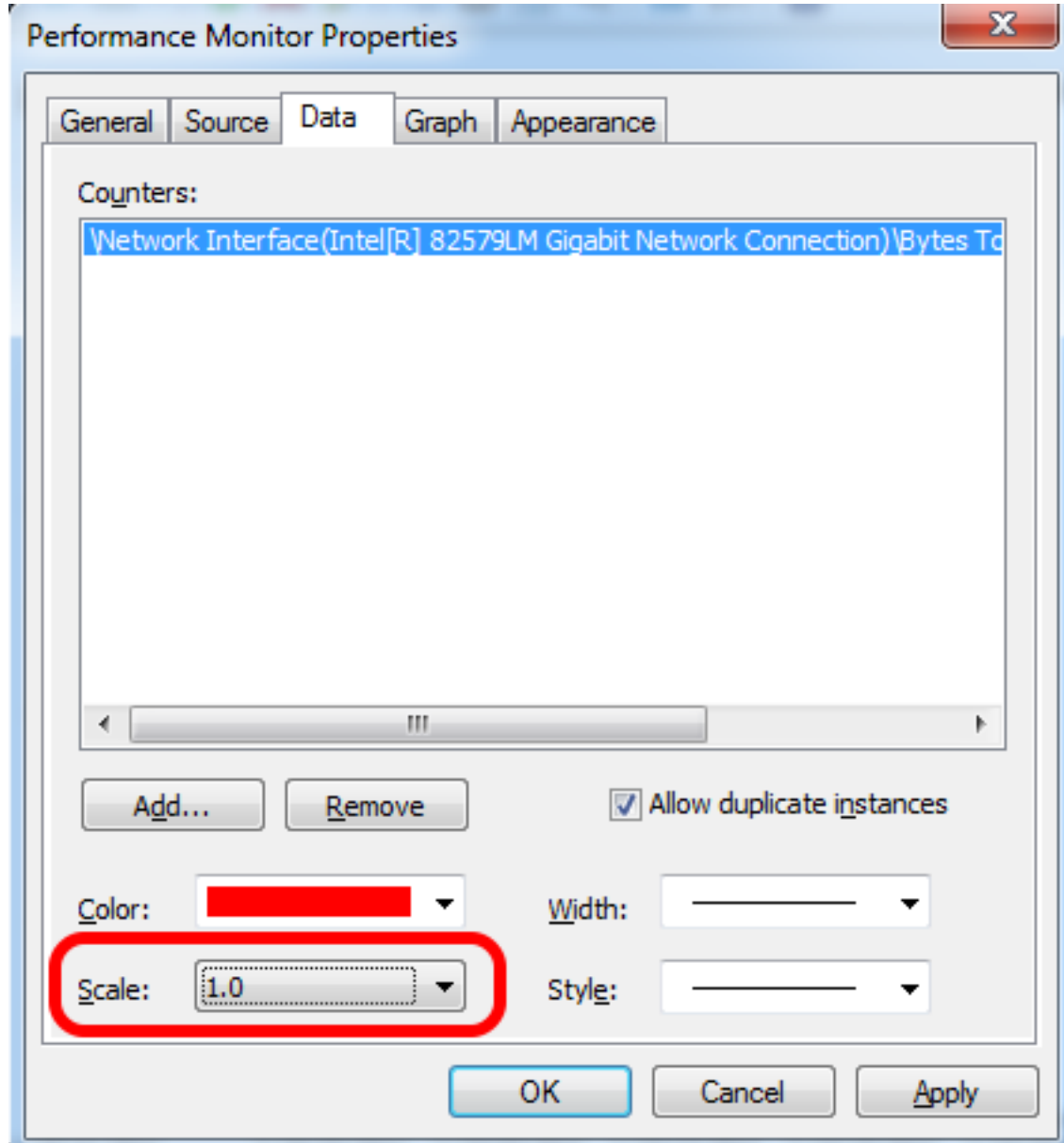
1. Click the green plus near the top of the screen to add a counter
2. In the top left pane, locate and click on Network Interface to select it
3. In the bottom left pane, locate the desired network interface (or use All instances to monitor all interfaces)
4. Click Add>> to add the counter to the right pane.
5. Click OK to add the counters to the graph.

Remove Extra Counters



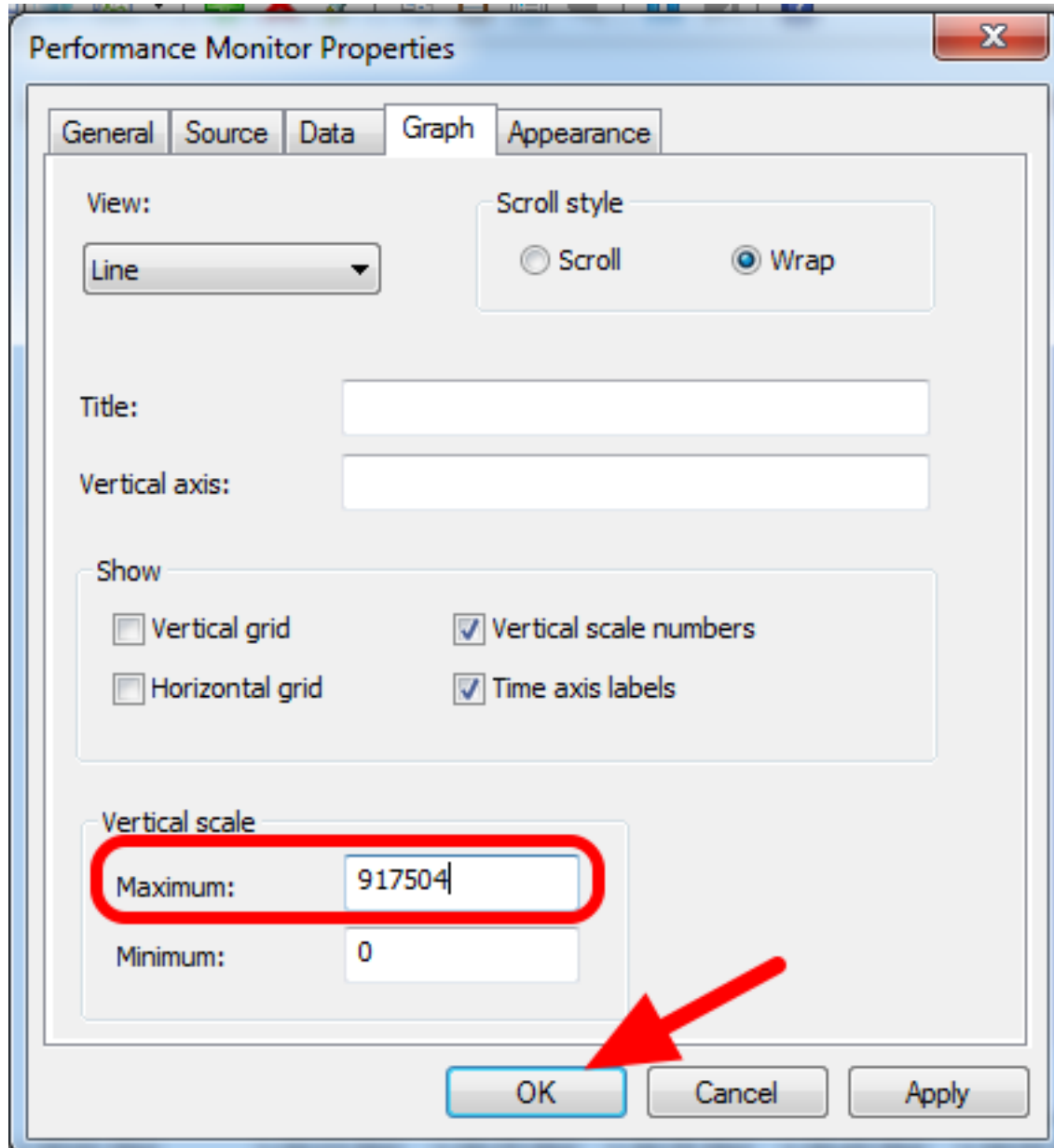
In the bottom pane, select each counter other than Bytes Total/sec and press the Delete key. The Bytes Total/sec entry should be the only entry remaining in the pane.

Configure Data Properties



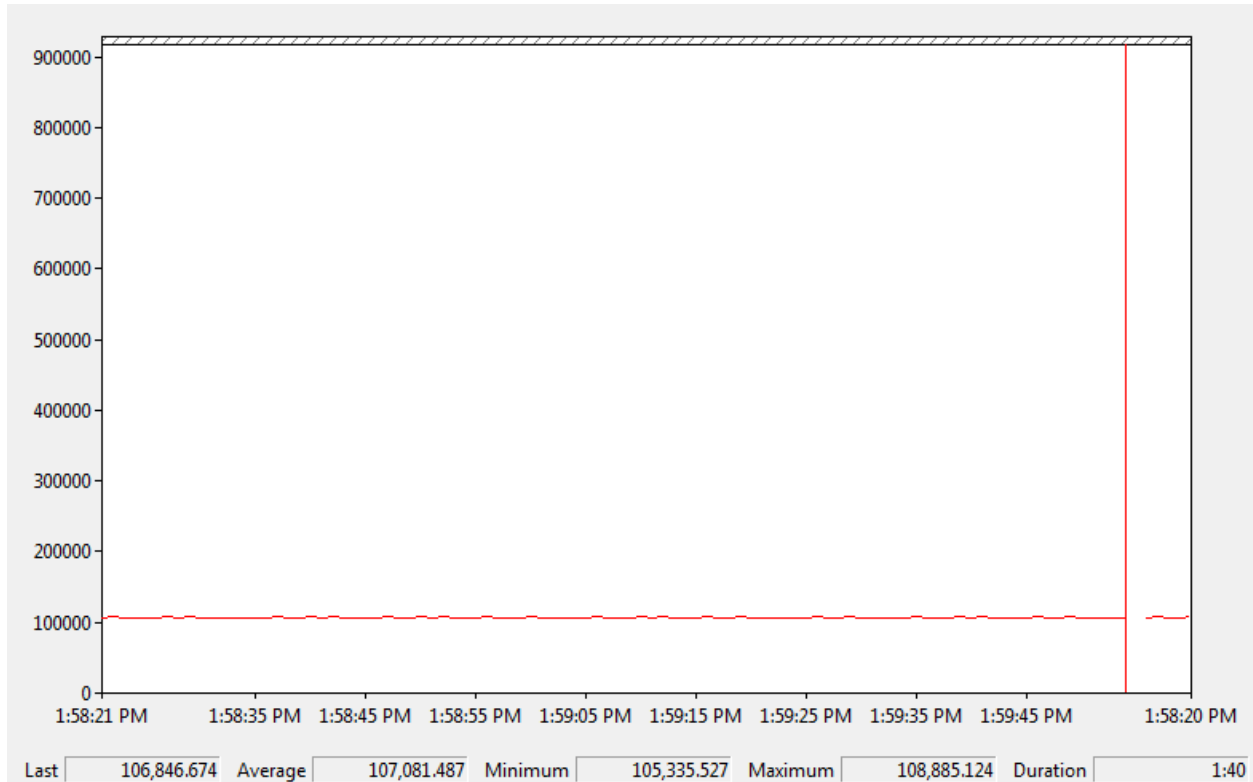
Press Ctrl+Q to bring up the Properties window. Click on the dropdown next to Scale and select 1.0. Then click on the Graph tab.

Configure Graph Properties



In the Maximum Box under Vertical Scale enter 917504 (this is 7 Megabits converted to Bytes). If desired, turn on the horizontal grid by checking the box. Then click OK to close the dialog.

Viewing Bandwidth Usage

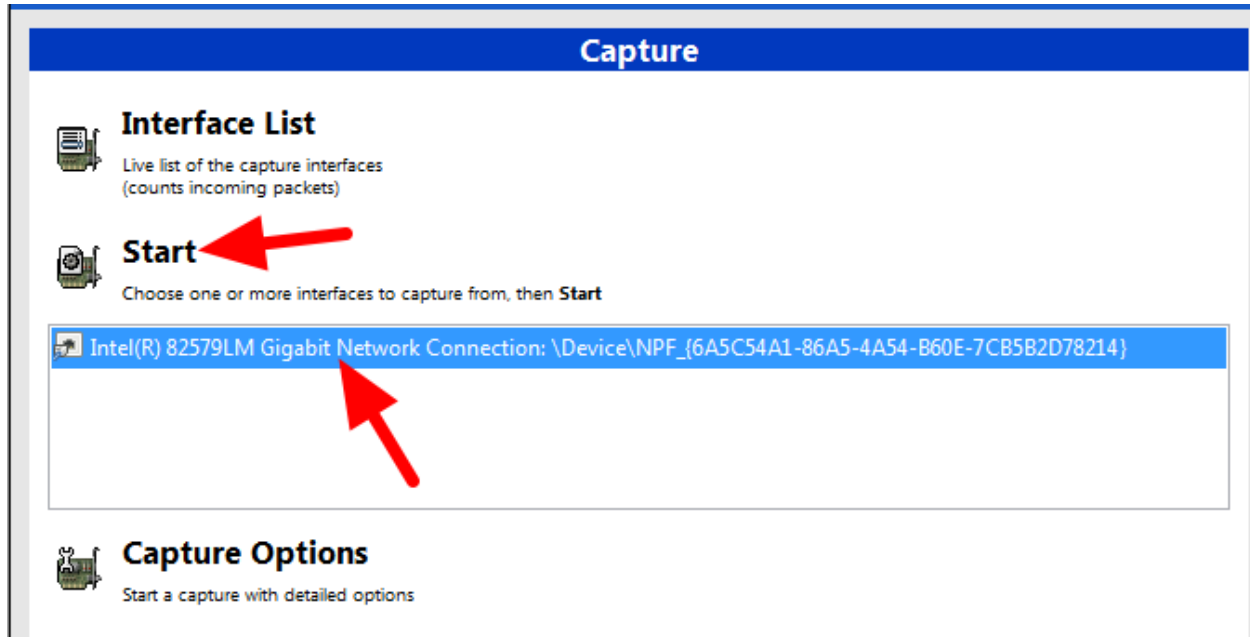


You may now connect to your robot as normal over the selected interface (if you haven't done so already). The graph will show the total bandwidth usage of the connection, with the bandwidth cap at the top of the graph. The Last, Average, Min and Max values are also displayed at the bottom of the graph. Note that these values are in Bytes/Second meaning the cap is 917,504. With just the Driver Station open you should see a flat line at ~100000 Bytes/Second.

21.5.2 Measuring Bandwidth Usage using Wireshark

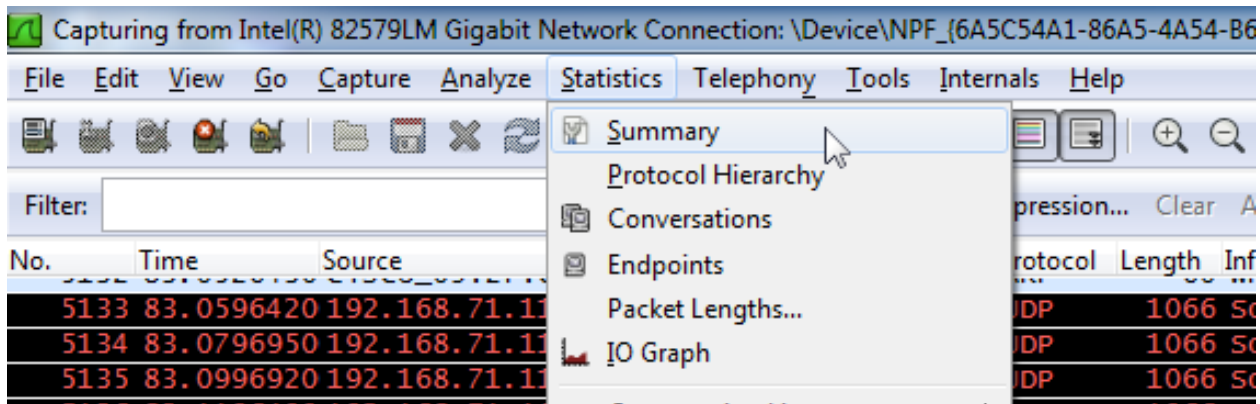
If you are not using Windows 7, you will need to install a 3rd party program to monitor bandwidth usage. One program that can be used for this purpose is Wireshark. Download and install the latest version of Wireshark for your version of Windows. After installation is complete, locate and open Wireshark. Connect your computer to your robot, open the Driver Station and any Dashboard or custom programs you may be using.

Select the interface and Start capture



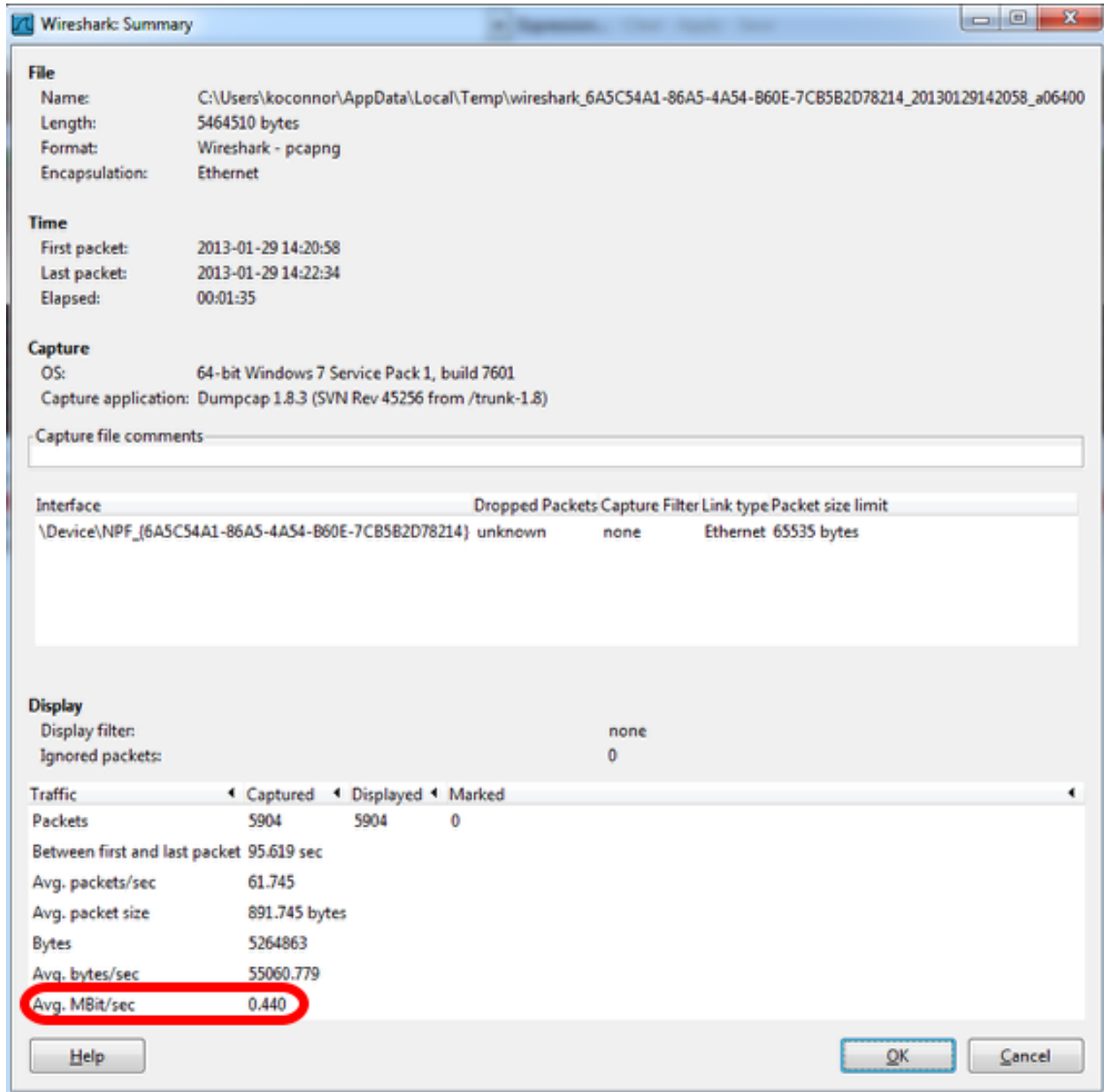
In the Wireshark program on the left side, select the interface you are using to connect to the robot and click Start.

Open Statistics Summary



Let the capture run for at least 1 minute, then click Statistics then Summary.

View Bandwidth Usage



Average bandwidth usage, in Megabits/Second is displayed near the bottom of the summary window.

21.6 OM5P-AC Radio Modification

The intended use case for the OM5P-AC radio does not subject it to the same shocks and forces as it sees in the FRC environment. If the radio is subjected to significant pressure on the bottom of the case, it is possible to cause a radio reboot by shorting a metal shield at the bottom of the radio to some exposed metal leads on the bottom of the board. This article details a modification to the radio to prevent this scenario.

Warning: It takes significant pressure applied to the bottom of the case to cause a reboot in this manner. Most FRC radio reboot issues can be traced to the power path in some form. We recommend mitigating this risk via strategic mounting of the radio rather than opening and modifying the radio (and risk damaging delicate internal components):

- Avoid using the “mounting tab” features on the bottom of the radio)
- You may wish to mount the radio to allow for some shock absorption. A little can go a long way, mounting the radio using hook and loop fastener or to a robot surface with a small amount of flex (plastic or sheet metal sheet, etc.) can significantly reduce the forces experienced by the radio.

21.6.1 Opening the Radio

Note: The OpenMesh OM5P-AC is not designed to be a user serviceable device. Users perform this modification at their own risk. Make sure to work slowly and carefully to avoid damaging internal components such as radio antenna cables.

Case Screws





Locate the two rubber feet on the front side of the radio then pry them off the radio using fingernails, small flat screwdriver, etc. Using a small Phillips screwdriver, remove the two screws under the feet.

Side Latches



There is a small latch on the lid of the radio near the middle of each long edge (you can see these latches more clearly in the next picture). Using a fingernail or very thin tool, slide along the gap between the lid and case from front to back towards the middle of the radio, you should hear a small pop as you near the middle of radio. Repeat on the other side (note: it's not hard to accidentally re-latch the first side while doing this, make sure both sides are unlatched before proceeding). The radio lid should now be slightly open on the front side as shown in the image above.

Remove Lid

Warning: The board may stick to the lid as you remove it due to the heatsink pads. Look through the vents of the radio as you remove the lid to see if the board is coming with it, if it is you may need to insert a small tool to hold the board down to separate it from the lid. We recommend a small screwdriver or similar tool that fits through the vents, applied through the front corner on the barrel jack side, right above the screw hole. You can scroll down to the picture with the lid removed to see what the board looks like in this area.

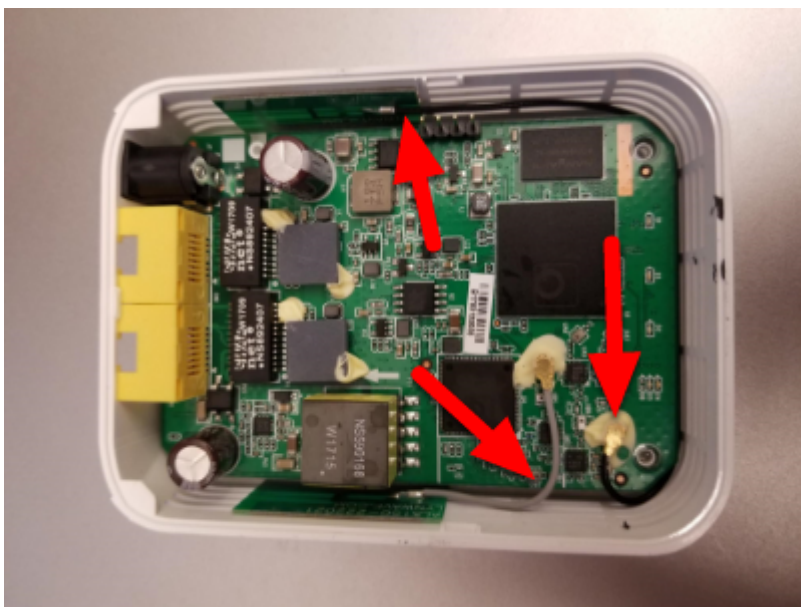


To begin removing the lid, slide it forward (lifting slightly) until the screw holders hit the case front (you may need to apply pressure on the latch areas while doing this).

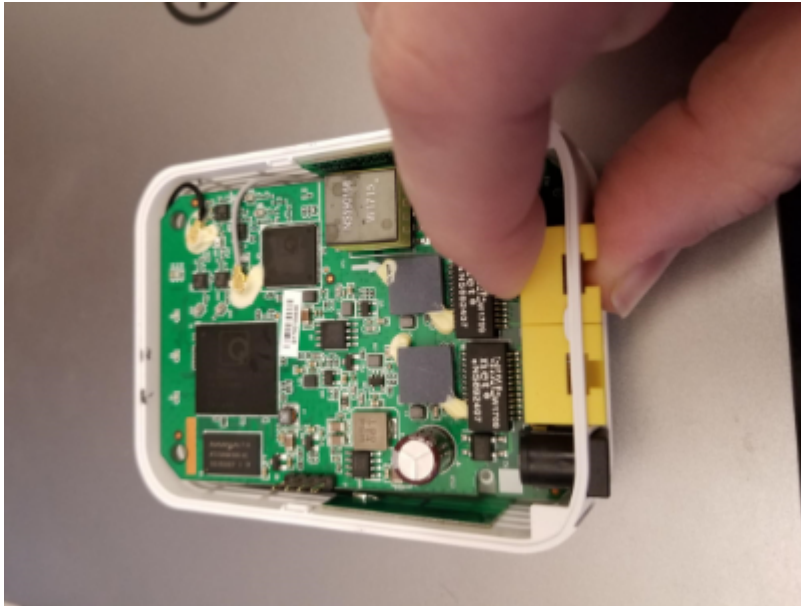


Next, begin rotating the lid slightly away from the barrel jack side, as shown while continuing to lift. This will unhook the lid from the small triangle visible in the top right corner. Continue to rotate slightly in this direction while pushing the top left corner towards the barrel jack (don't try to lift further in this step) to unhook a similar feature in the top left corner. Then lift the lid completely away from the body.

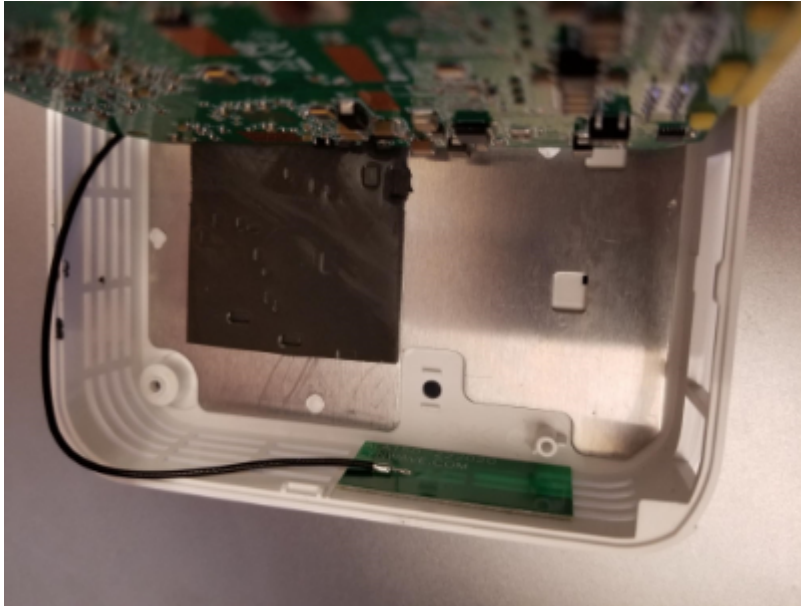
Remove Board



Warning: Note the antenna wires shown in the image above. These wires, and their connectors, are fragile, take care not to damage them while performing the next steps.



To remove the board, we recommend grasping one or both network ports with your fingers (as shown) and pushing inward (toward the front of the radio) and upward until the network ports and barrel jack are free from the case.



Tilt the board up (towards the short grey antenna cable) to expose the metal shield underneath.

Note: When you perform this step, you may notice that there is a small reset button on the underside of the board that is larger than the hole in the case. Note that pressing the reset button with the FRC firmware installed has no effect and that drilling the case of the radio is not a permitted modification.

21.6.2 Apply Tape



Apply a piece of electrical tape to the metal shield in the area just inside of the network port/barrel jack openings. This will prevent the exposed leads on the underside of the board

from short circuiting on this plate.

21.6.3 Re-assemble Radio

Re-assemble the radio by reversing the instructions to open it:

- Lay the board back down, making sure it aligns with the screw holes near the front and seats securely
- Slide the lid onto the back left retaining feature by moving it in from right to left. Take care of the capacitor in this area
- Rotate the lid, press downwards and slide the back right retaining feature in
- Press down firmly on the front/middle of the lid to seat the latches
- Replace 2 screws in front feet
- Replace front feet

Entre em contato conosco: robotica.1156@gmail.com