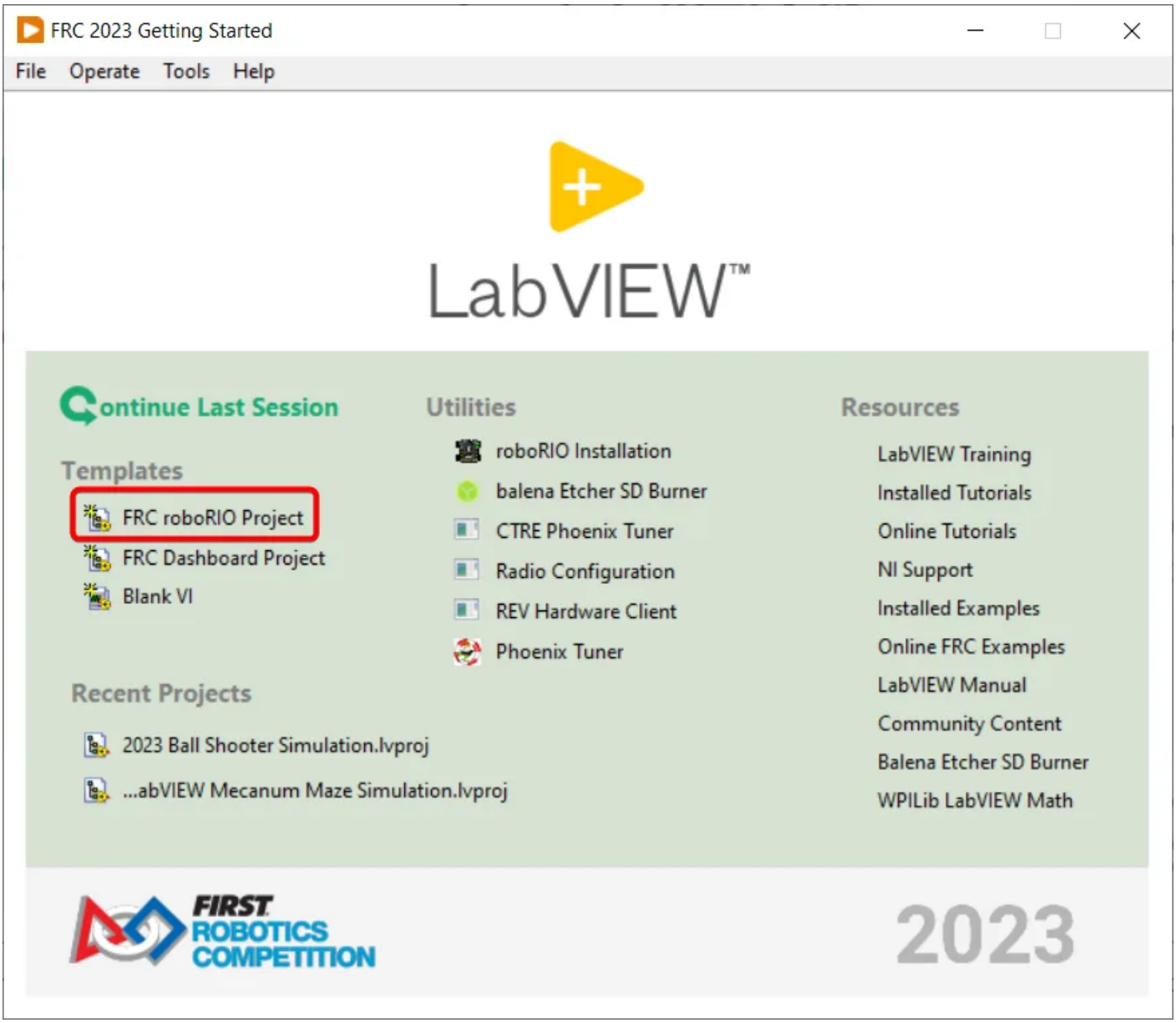
Etapa 4: Programando seu robô

- Criando seu programa de teste para tração (LabVIEW)
- Criando seu programa de teste (C++/Java)
- Programando seu robô (C++)
- Executando seu programa de teste

Criando seu programa de teste para tração (LabVIEW)

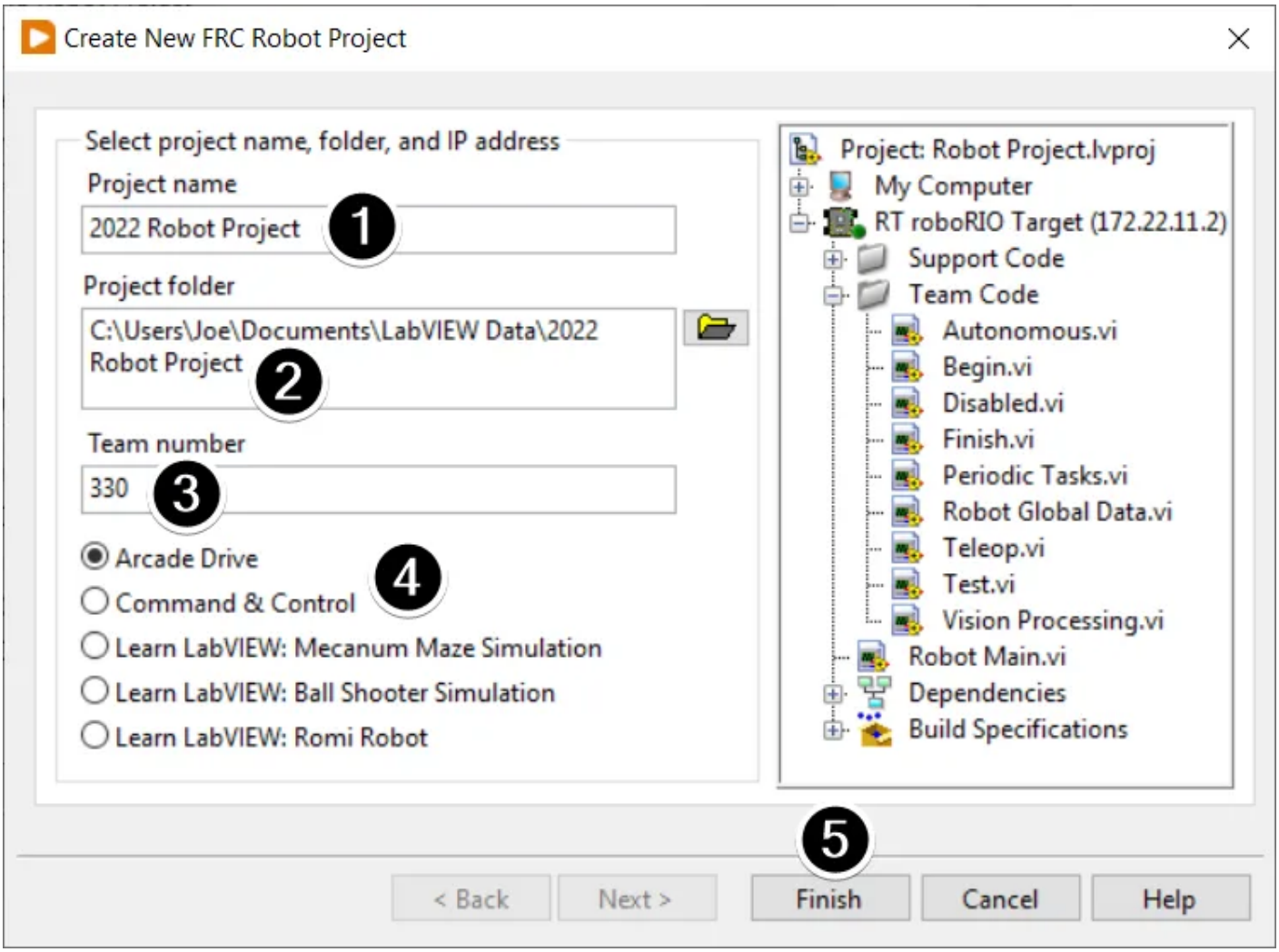
> Esse documento aborda como criar, construir e carregar uma programação básica de FRC® LabVIEW para uma tração no roboRIO. Antes de começar, tenha certeza que você instalou o LabVIEW para FR, as FRC Game Tools e que você configurou seu roboRIO como descrito no tutorial Do zero ao robô.

Criando um projeto



Inicie o LabVIEW e clique no *FRC roboRIO Robot Project* para abrir a caixa de dialógo *Create New FRC Robot Project*.

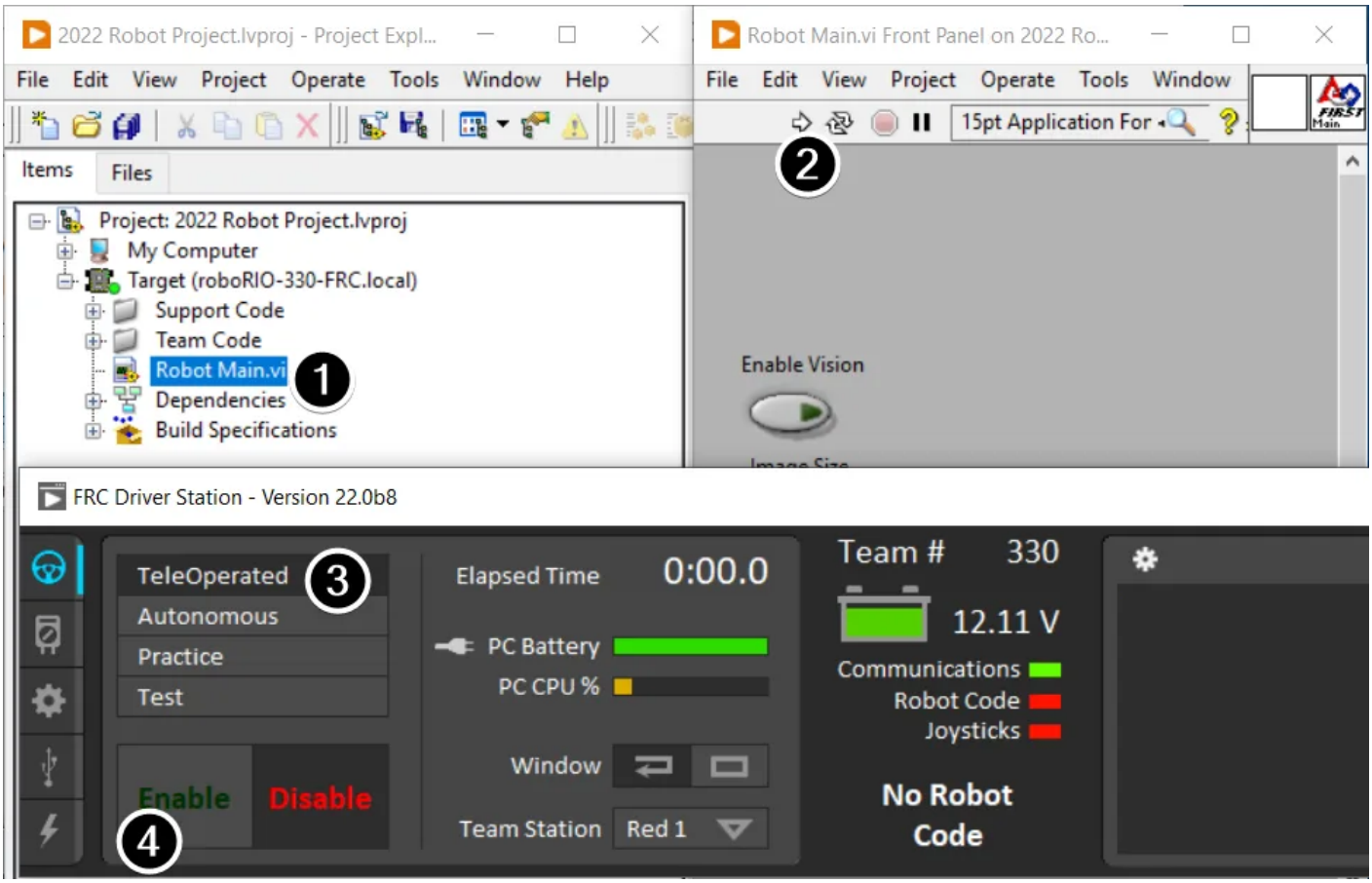
Configurando o projeto



Preencha a caixa de dialógo *Crate New FRC Robot Project*

1. Escolha um nome para seu projeto
2. Selecione uma pasta para colocar o projeto
3. Coloque o número do seu time
4. Selecione um tipo de projeto. Se não tiver certeza, selecione *Arcade Drive*
5. Clique em *Finish*

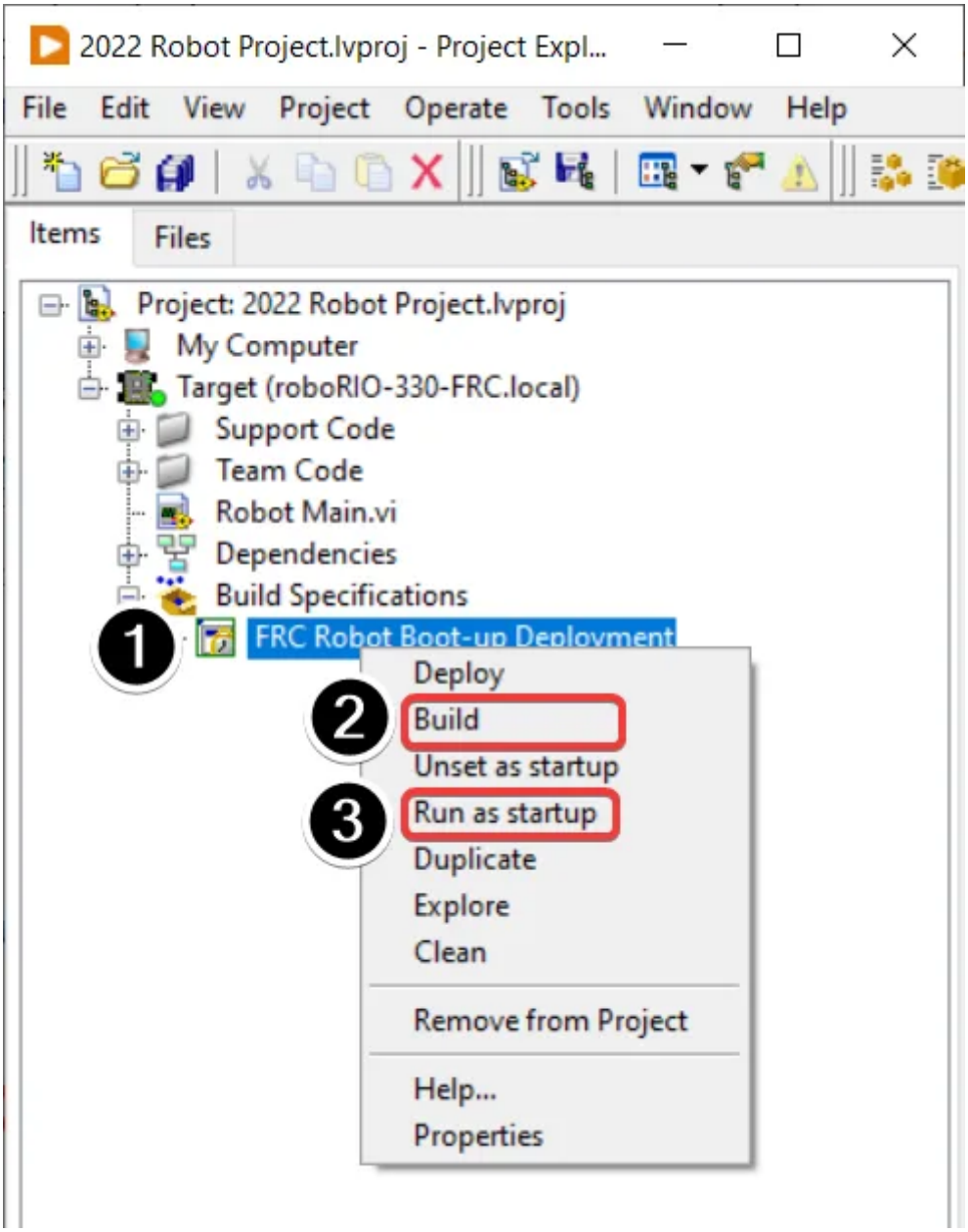
Executando o programa



> Nota-se que o programa executado dessa maneira não estara no roboRIO depois de um ciclo de ligação. Para implementar um programa para executar toda vez que o roboRIO inícia, siga a próxima etapa, implementando o programa.

1. Na janela de explorador de projeto, clique duas vezes em Robot Main.vi para abrir o Robot Main.VI
2. Clique no botão de *Run* (Flecha branca na aba superior) do Robot Main.VI para implementar o VI no roboRIO. LaVIEW implementa o VI, todos os items exigidos pelo VI, e as configurações alvo na memória do roboRIO. Se perguntar para salvar qualquer VIs, clique em *Save on all prompts*.
3. Usando o programa da Driver Stations, coloque o robô no modo teleoperado. Para mais informações na configuração e utilização da Driver Station, veja o artigo *FRC Driver Stations Software*.
4. Clique em *Enable*
5. Mova o joystick e observe como o robô responde.
6. Clique no *Abort button* do Robot Main.VI. Observe que o VI para. Quando você executa um programa utilizando o botão de *Run*, o programa opera no roboRIO, mas você consegue manipular os objetos no *front panel* pelo computador em que o programa esta sendo executado.

Implementando o programa



Para operar na competição, você vai precisar implementar o robô no seu roboRIO. Isso permite que o programa sobreviva a *resets* do controlador, mas não permite as mesmas ferramentas de depuração (front panel, probes highlight execution) que estão operando no *front panel*. Para implementar seu programa:

1. No explorador de projetos clique no + perto de *Build Specifications* para expandi-lo.
2. Botão direito em *FRC Robot Boot-up Deployment* e selecione *Build*. Espere pela implementação para terminar.
3. Botão direito novamente em *FRC Robot Boot-Up Deployment* e selecione *Run as Startup*. Se você receber um dialógo de conflito, cique em *OK*. Esse dialógo indica que o programa atual no roboRIO vai ser terminado/substituído.

4. Either check the box to close the deployment window on successful completion ou clique no botão *close* quando a implementação for um sucesso.
5. O roboRIO vai automáticamente começar operando o código implantado dentro de poucos segundos após a caixa de dialógo fechar.

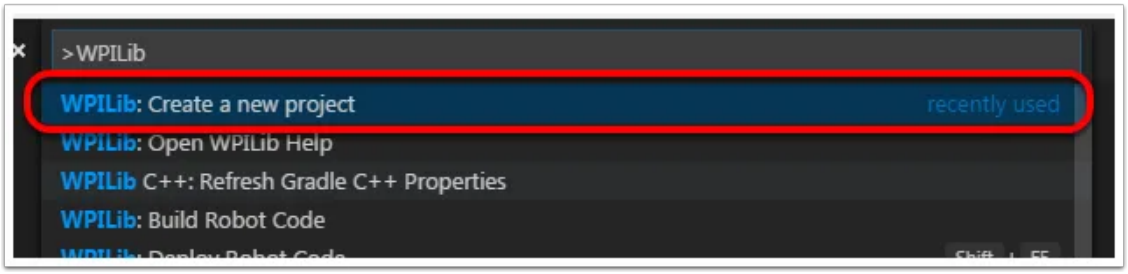
Criando seu programa de teste (C++/Java)

Assim que tudo estiver instalado, estamos prontos para criar um programa de robô. O WPILib vem com vários modelos para programas de robô. O uso desses modelos é altamente recomendado para novos usuários; no entanto, usuários avançados têm a liberdade de escrever seu próprio código de robô do zero. Este artigo guia você na criação de um projeto a partir de um dos exemplos fornecidos, que já possui algum código escrito para controlar um robô básico.

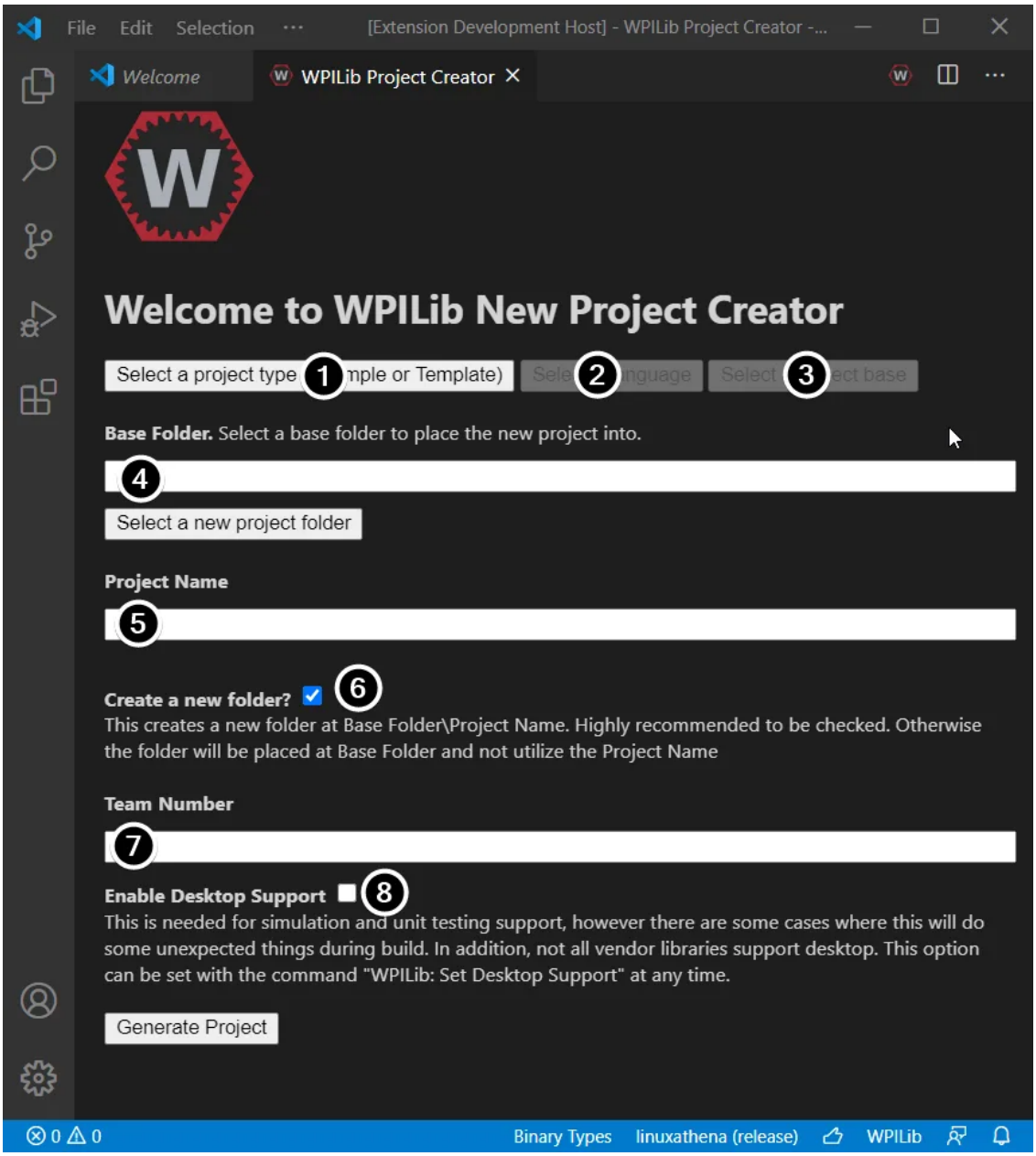
> Este guia inclui exemplos de código que envolvem hardware do fornecedor para a conveniência do usuário. Neste documento, PWM refere-se ao controlador de motor incluído no KOP. A guia CTRE faz referência ao controlador de motor Talon FX (motor Falcon 500), mas o uso é semelhante para TalonSRX e VictorSPX. A guia REV faz referência ao CAN SPARK MAX controlando um motor sem escova, mas é semelhante para motor com escovas. Parte-se do pressuposto de que o usuário já instalou os vendordeps necessários e configurou o(s) dispositivo(s) (atualização de firmware, atribuição de IDs CAN, etc.) de acordo com a documentação do fabricante (CTRE REV).

Criando um novo projeto WPILib

Abra a paleta de comandos do Visual Studio Code com Ctrl+Shift+P. Em seguida, digite "WPILib" no prompt. Como todos os comandos do WPILib começam com "WPILib", isso mostrará a lista de comandos específicos do WPILib no VS Code. Agora, selecione o comando "Criar um novo projeto".



Isso abrirá a "Janela do Criador de Novo Projeto:".



Os elementos da "Janela do Criador de Novo Projeto" são explicados abaixo:

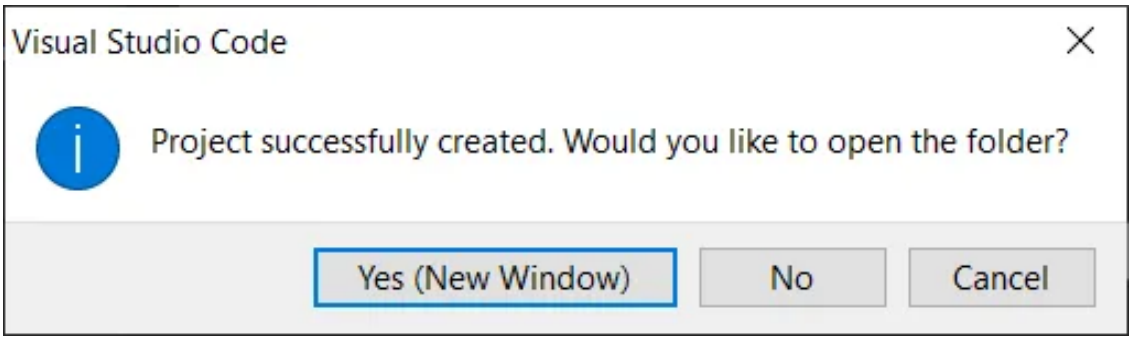
1. Tipo de Projeto: O tipo de projeto que desejamos criar. Para este exemplo, selecione "Exemplo".
2. Linguagem: Esta é a linguagem (C++ ou Java) que será usada para este projeto.
3. Base do Projeto: Esta caixa é usada para selecionar a classe base ou exemplo a ser gerado para o projeto. Para este exemplo, selecione "Início Rápido" (Getting Started).
4. Pasta Base: Isso determina a pasta na qual o projeto do robô será localizado.
5. Nome do Projeto: O nome do projeto do robô. Isso também especifica o nome que será dado à pasta do projeto se a caixa "Criar Nova Pasta" estiver marcada.
6. Criar Nova Pasta: Se esta opção estiver marcada, será criada uma nova pasta para armazenar o projeto dentro da pasta especificada anteriormente. Se não estiver marcada, o projeto será localizado diretamente na pasta especificada anteriormente. Um erro será gerado se a pasta não estiver vazia e esta opção não estiver marcada.
7. Número da Equipe: O número da equipe para o projeto, que será usado para os nomes de pacotes dentro do projeto e para localizar o robô ao implantar o código.

8. Habilitar Suporte de Desktop: Ativa os testes de unidade e a simulação. Embora o WPILib suporte isso, bibliotecas de software de terceiros podem não oferecer suporte a isso. Se as bibliotecas não suportarem desktop, seu código pode não compilar ou pode travar. Deve ser deixado desmarcado, a menos que os testes de unidade ou a simulação sejam necessários e todas as bibliotecas o suportem. Para este exemplo, não marque esta caixa.

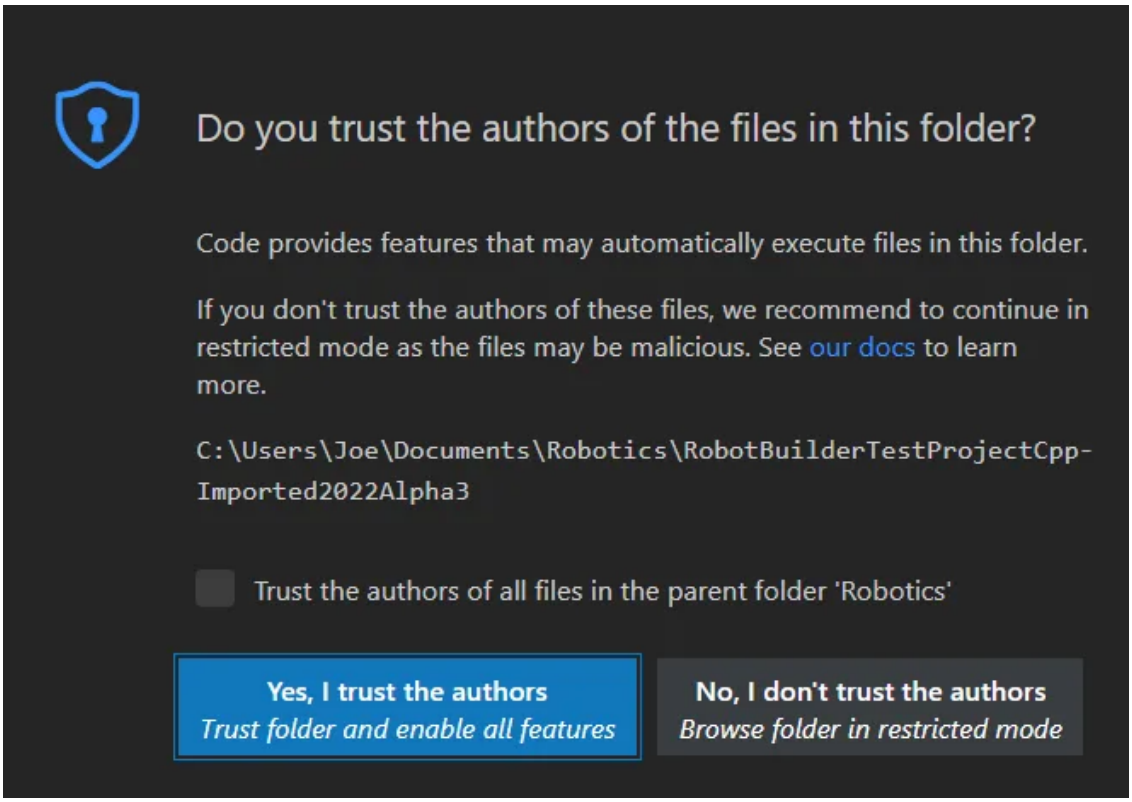
Depois de configurar todos os itens acima, clique em "Gerar Projeto" e o projeto do robô será criado.

> Quaisquer erros na geração do projeto aparecerão no canto inferior direito da tela.

Abrindo o novo projeto

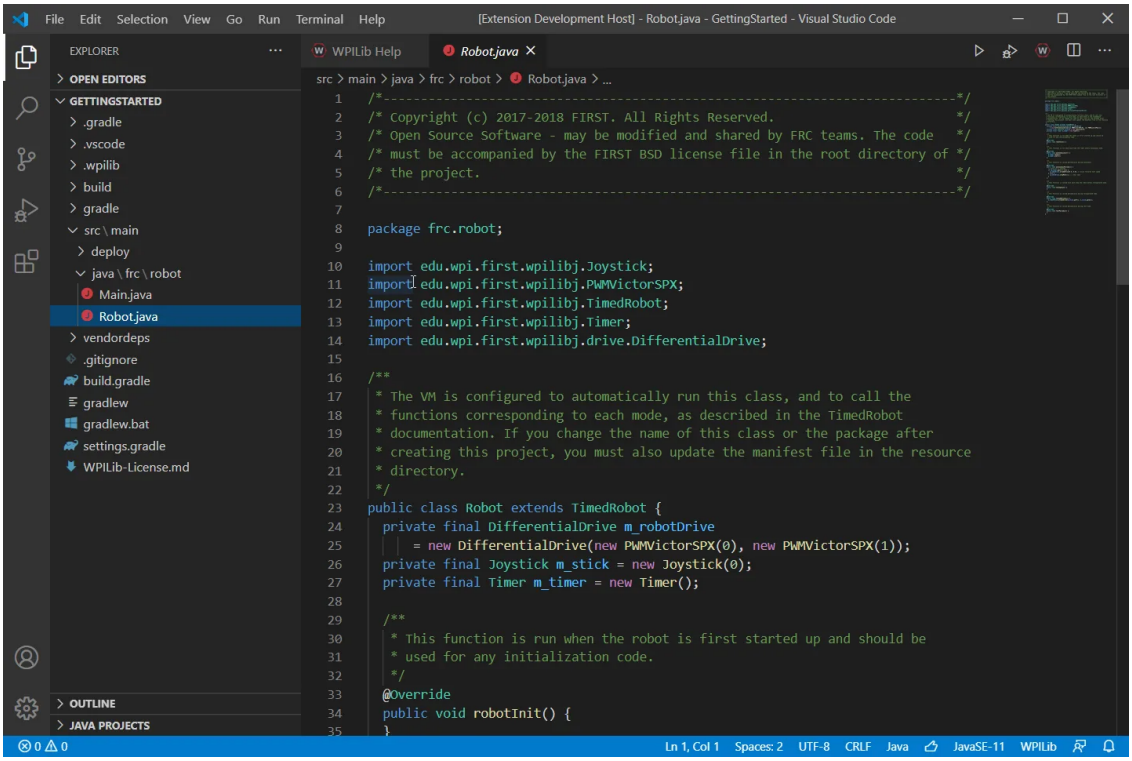


Após criar com sucesso o seu projeto, o VS Code dará a opção de abrir o projeto, conforme mostrado acima. Podemos optar por fazer isso agora ou mais tarde, digitando Ctrl+K e, em seguida, Ctrl+O (ou apenas Command+O no macOS) e selecionar a pasta onde salvamos nosso projeto.



Clique em "Sim, confio nos autores".

Depois de abrir, veremos a hierarquia do projeto à esquerda. Clicar duas vezes no arquivo abrirá esse arquivo no editor.



Importações

PWM

```
import edu.wpi.first.wpilibj.TimedRobot;
import edu.wpi.first.wpilibj.Timer;
import edu.wpi.first.wpilibj.XboxController;
import edu.wpi.first.wpilibj.drive.DifferentialDrive;
import edu.wpi.first.wpilibj.motorcontrol.PWMSparkMax;
```

CTRE

```
import edu.wpi.first.wpilibj.Joystick;
import edu.wpi.first.wpilibj.TimedRobot;
import edu.wpi.first.wpilibj.Timer;
import edu.wpi.first.wpilibj.drive.DifferentialDrive;
import com.ctre.phoenix.motorcontrol.can.WPI_TalonFX;
```

REV

```
import com.revrobotics.CANSparkMax;
import com.revrobotics.CANSparkMaxLowLevel.MotorType;

import edu.wpi.first.wpilibj.TimedRobot;
import edu.wpi.first.wpilibj.Timer;
import edu.wpi.first.wpilibj.XboxController;
import edu.wpi.first.wpilibj.drive.DifferentialDrive;
```

Nosso código precisa fazer referência aos componentes do WPILib que estão sendo utilizados. Em Java, é feito com declarações import. O programa faz referência a classes como Joystick (para direção), PWMSparkMax / WPI_TalonFX / CANSparkMax (para controle de motores), TimedRobot (a classe base usada no exemplo), Timer (usado para o modo autônomo) e DifferentialDrive (para conectar o controle do joystick aos motores).

Definindo as variáveis para nosso robô de exemplo

PWM

```
public class Robot extends TimedRobot {
  private final PWMSparkMax m_leftDrive = new PWMSparkMax(0);
  private final PWMSparkMax m_rightDrive = new PWMSparkMax(1);
  private final DifferentialDrive m_robotDrive = new DifferentialDrive(m_leftDrive,
m_rightDrive);
  private final XboxController m_controller = new XboxController(0);
  private final Timer m_timer = new Timer();

  /**
   * This function is run when the robot is first started up and should be used for any
   * initialization code.
   */
  @Override
  public void robotInit() {
    // We need to invert one side of the drivetrain so that positive voltages
    // result in both sides moving forward. Depending on how your robot's
    // gearbox is constructed, you might have to invert the left side instead.
    m_rightDrive.setInverted(true);
  }
```

CTRE

```
public class Robot extends TimedRobot {
   private final WPI_TalonFX m_leftDrive = new WPI_TalonFX(1);
   private final WPI_TalonFX m_rightDrive = new WPI_TalonFX(2);
   private final DifferentialDrive m_robotDrive = new DifferentialDrive(m_leftDrive,
m_rightDrive);
   private final Joystick m_stick = new Joystick(0);
   private final Timer m_timer = new Timer();
```

REV

```
public class Robot extends TimedRobot {
  private final CANSparkMax m_leftDrive = new CANSparkMax(1, MotorType.kBrushless);
  private final CANSparkMax m_rightDrive = new CANSparkMax(2, MotorType.kBrushless);
  private final DifferentialDrive m_robotDrive = new DifferentialDrive(m_leftDrive,
m_rightDrive);
  private final XboxController m_controller = new XboxController(0);
  private final Timer m_timer = new Timer();
```

O robô de exemplo em nossos exemplos terá um joystick na porta USB 0 para controle tipo arcade e dois motores nas portas PWM 0 e 1 (exemplos do fornecedor usam CAN com IDs 1 e 2). Aqui, criamos objetos do tipo DifferentialDrive (m_robotDrive), Joystick (m_stick) e Timer (m_timer). Esta seção do código faz três coisas:

1. Define as variáveis como membros da nossa classe Robot.
2. Inicializa as variáveis.

Inicialização do robô

```
@Override
public void robotInit() {}
```

O método RobotInit é executado quando o programa do robô está iniciando, mas após o construtor. O RobotInit para o nosso programa de exemplo não faz nada. Se quisermos executar algo aqui, podemos fornecer o código acima para substituir o padrão.

Autônomo simples de exemplo

```
  /** This function is run once each time the robot enters autonomous mode. */
  @Override
  public void autonomousInit() {
    m_timer.restart();
  }

  /** This function is called periodically during autonomous. */
  @Override
```

```
public void autonomousPeriodic() {
  // Drive for 2 seconds
  if (m_timer.get() < 2.0) {
    // Drive forwards half speed, make sure to turn input squaring off
    m_robotDrive.arcadeDrive(0.5, 0.0, false);
  } else {
    m_robotDrive.stopMotor(); // stop robot
  }
}
```

O método AutonomousInit é executado uma vez cada vez que o robô faz a transição para o modo autônomo a partir de outro modo. Neste programa, reiniciamos o Timer neste método.

AutonomousPeriodic é executado uma vez a cada período enquanto o robô está no modo autônomo. Na classe TimedRobot, o período é um tempo fixo, que padrão é 20ms. Neste exemplo, o código periódico verifica se o temporizador é inferior a 2 segundos e, se for, avança a meio velocidade usando o método ArcadeDrive da classe DifferentialDrive. Se mais de 2 segundos tiverem decorrido, o código para o acionamento do robô.

Controle por analógico para teleoperado

```
/** This function is called once each time the robot enters teleoperated mode. */
@Override
public void teleopInit() {}

/** This function is called periodically during teleoperated mode. */
@Override
public void teleopPeriodic() {
  m_robotDrive.arcadeDrive(-m_controller.getLeftY(), -m_controller.getRightX());
}
```

Assim como no modo Autônomo, o modo Teleop possui funções TeleopInit e TeleopPeriodic. Neste exemplo, não temos nada para fazer em TeleopInit; ele é fornecido apenas para fins ilustrativos. Em TeleopPeriodic, o código utiliza o método ArcadeDrive para mapear o eixo Y do Joystick para o movimento para frente/trás dos motores de acionamento e o eixo X para o movimento de virar.

Modo de teste

```
/** This function is called once each time the robot enters test mode. */
@Override
public void testInit() {}

/** This function is called periodically during test mode. */
@Override
public void testPeriodic() {}
```

O Modo de Teste é usado para testar a funcionalidade do robô. Semelhante ao TeleopInit, os métodos TestInit e TestPeriodic são fornecidos aqui apenas para fins ilustrativos.

Compilando o projeto no robô

Por favor, consulte as instruções aqui para implantar o programa em um robô.

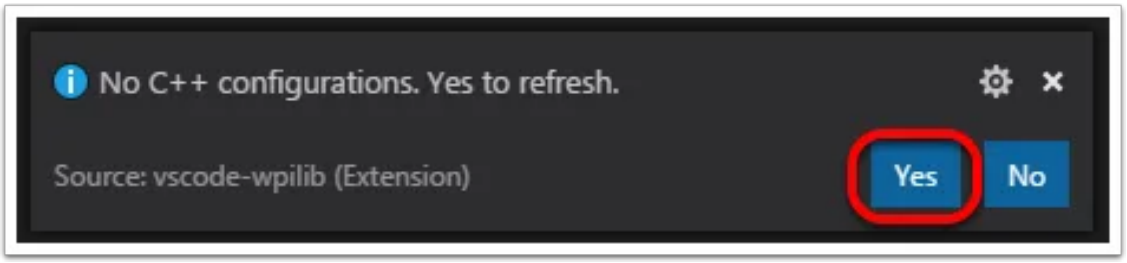
Programando seu robô (C++)

Essa página oferece para usuários de C++ um ensinamento básico de programação para FRC

Ainda que você não esteja usando Java, lembre-se de olhar a página anterior para criar seu projeto de robô

Configurações C++

Para projetos em C++, há mais uma etapa para configurar o IntelliSense. Sempre que abrimos um projeto, deveríamos receber um pop-up no canto inferior direito perguntando se desejamos atualizar as configurações do C++. Clique em "Sim" para configurar o IntelliSense.



Importações

PWM

```
#include <frc/TimedRobot.h>
#include <frc/Timer.h>
#include <frc/XboxController.h>
#include <frc/drive/DifferentialDrive.h>
```

```
#include <frc/motorcontrol/PWMSparkMax.h>
```

CTRE

```
#include <frc/Joystick.h>
#include <frc/TimedRobot.h>
#include <frc/Timer.h>
#include <frc/drive/DifferentialDrive.h>
#include <ctre/phoenix/motorcontrol/can/WPI_TalonFX.h>
```

REV

```
#include <frc/TimedRobot.h>
#include <frc/Timer.h>
#include <frc/XboxController.h>
#include <frc/drive/DifferentialDrive.h>
#include <frc/motorcontrol/PWMSparkMax.h>

#include <rev/CANSparkMax.h>
```

Nosso código precisa fazer referência aos componentes do WPILib que estão sendo utilizados. Em C++, isso é feito usando declarações #include;. O programa faz referência a classes como Joystick (para direção), PWMSparkMax / WPI_TalonFX / CANSparkMax (para controle de motores), TimedRobot (a classe base usada no exemplo), Timer (usado para o modo autônomo) e DifferentialDrive (para conectar o controle do joystick aos motores).

Definindo as variáveis para nosso robô de exemplo

PWM

```
public:
 Robot() {
   // We need to invert one side of the drivetrain so that positive voltages
```

```
    // result in both sides moving forward. Depending on how your robot's
    // gearbox is constructed, you might have to invert the left side instead.
    m_right.SetInverted(true);
    m_robotDrive.SetExpiration(100_ms);
    m_timer.Start();
  }
```

```
 private:
  // Robot drive system
  frc::PWMSparkMax m_left{0};
  frc::PWMSparkMax m_right{1};
  frc::DifferentialDrive m_robotDrive{m_left, m_right};

  frc::XboxController m_controller{0};
  frc::Timer m_timer;
};
```

CTRE

```
public:
 Robot() {
    m_right.SetInverted(true);
    m_robotDrive.SetExpiration(100_ms);
    // We need to invert one side of the drivetrain so that positive voltages
    // result in both sides moving forward. Depending on how your robot's
    // gearbox is constructed, you might have to invert the left side instead.
    m_timer.Start();
 }
```

```
private:
 // Robot drive system
 ctre::phoenix::motorcontrol::can::WPI_TalonFX m_left{1};
 ctre::phoenix::motorcontrol::can::WPI_TalonFX m_right{2};
 frc::DifferentialDrive m_robotDrive{m_left, m_right};

 frc::Joystick m_stick{0};
 frc::Timer m_timer;
```

REV

```
Robot() {
  // We need to invert one side of the drivetrain so that positive voltages
  // result in both sides moving forward. Depending on how your robot's
  // gearbox is constructed, you might have to invert the left side instead.
  m_right.SetInverted(true);
  m_robotDrive.SetExpiration(100_ms);
  m_timer.Start();
}
```

```
private:
 // Robot drive system
 rev::CANSparkMax m_left{1, rev::CANSparkMax::MotorType::kBrushless};
 rev::CANSparkMax m_right{2, rev::CANSparkMax::MotorType::kBrushless};
 frc::DifferentialDrive m_robotDrive{m_left, m_right};

 frc::XboxController m_controller{0};
 frc::Timer m_timer;
```

O robô de exemplo em nossos exemplos terá um joystick na porta USB 0 para direção arcade e dois motores nas portas PWM 0 e 1 (exemplos do fornecedor usam CAN com IDs 1 e 2). Aqui criamos objetos do tipo DifferentialDrive (m_robotDrive), Joystick (m_stick) e Timer (m_timer). Esta seção do código faz duas coisas:

1. Define as variáveis como membros de nossa classe Robot.
2. Inicializa as variáveis.

> As inicializações de variáveis para C++ estão na seção privada na parte inferior do programa. Isso significa que elas são privadas para a classe (Robot). O código em C++ também define o tempo de expiração do Motor Safety para 0,1 segundos (o acionamento será desligado se não receber um comando a cada 0,1 segundos) e inicia o Timer usado para o modo autônomo.

Inicialização do robô

```
void RobotInit() {}
```

O método RobotInit é executado quando o programa do robô está inicializando, mas após o construtor. O RobotInit para o nosso programa de exemplo não faz nada. Se quisermos executar alguma coisa aqui, poderíamos fornecer o código acima para substituir o padrão.

Autônomo simples de exemplo

```
void AutonomousInit() override { m_timer.Restart(); }

void AutonomousPeriodic() override {
  // Drive for 2 seconds
  if (m_timer.Get() < 2_s) {
    // Drive forwards half speed, make sure to turn input squaring off
    m_robotDrive.ArcadeDrive(0.5, 0.0, false);
  } else {
    // Stop robot
    m_robotDrive.ArcadeDrive(0.0, 0.0, false);
  }
}
```

O método AutonomousInit é executado uma vez cada vez que o robô faz a transição para o modo autônomo a partir de outro modo. Neste programa, reiniciamos o Timer neste método.

AutonomousPeriodic é executado uma vez a cada período enquanto o robô está no modo autônomo. Na classe TimedRobot, o período é um tempo fixo, que por padrão é de 20ms. Neste exemplo, o código periódico verifica se o temporizador é inferior a 2 segundos e, se for o caso, move-se para frente a meia velocidade usando o método ArcadeDrive da classe DifferentialDrive. Se mais de 2 segundos tiverem se passado, o código para o acionamento do robô.

Controle de analógico para teleoperado

```
void TeleopInit() override {}

void TeleopPeriodic() override {
  // Drive with arcade style (use right stick to steer)
  m_robotDrive.ArcadeDrive(-m_controller.GetLeftY(),
```

```
                            m_controller.GetRightX());
  }
```

Assim como no modo Autônomo, o modo Teleop possui funções TeleopInit e TeleopPeriodic. Neste exemplo, não temos nada a fazer em TeleopInit; ele é fornecido apenas para fins ilustrativos. Em TeleopPeriodic, o código utiliza o método ArcadeDrive para mapear o eixo Y do joystick no movimento para frente/trás dos motores de acionamento e o eixo X no movimento de viragem.

Modo de teste

```
  void TestInit() override {}


  void TestPeriodic() override {}
```

O Modo de Teste é utilizado para testar a funcionalidade do robô. Semelhante ao TeleopInit, os métodos TestInit e TestPeriodic são fornecidos aqui apenas para fins ilustrativos.

Compilando o projeto no robô

Por favor, consulte as instruções aqui para implantar o programa em um robô.

Executando seu programa de teste

Visão geral

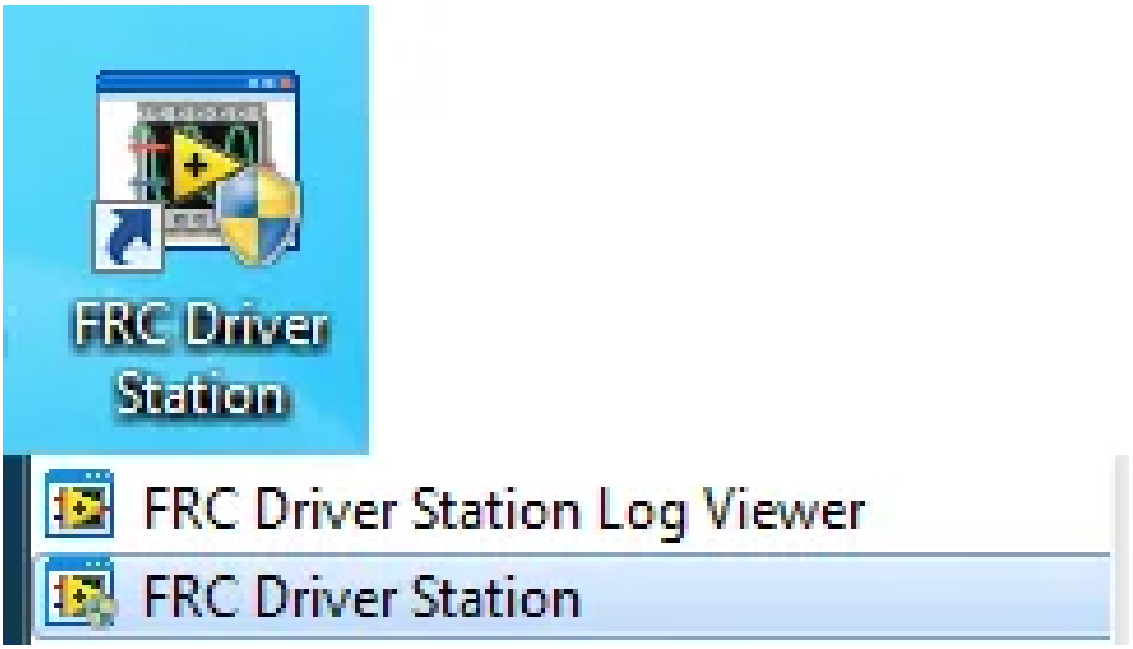
Você deve criar e instalar um programa de teste para sua linguagem escolhida, como visto nas seções anteriores.

Operação amarrada

Executar o seu programa de teste enquanto conectado à Estação do Condutor via cabo Ethernet ou USB confirmará que o programa foi implantado com sucesso e que a estação do condutor e o roboRIO estão configurados corretamente.

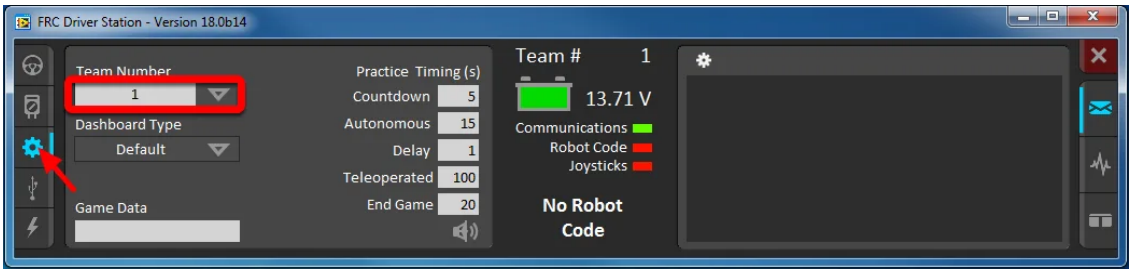
O roboRIO deve estar ligado e conectado ao PC através de Ethernet ou USB.

Iniciando a FRC Driver Station



Clique duas vezes no atalho para iniciar

Configurando a Driver Station

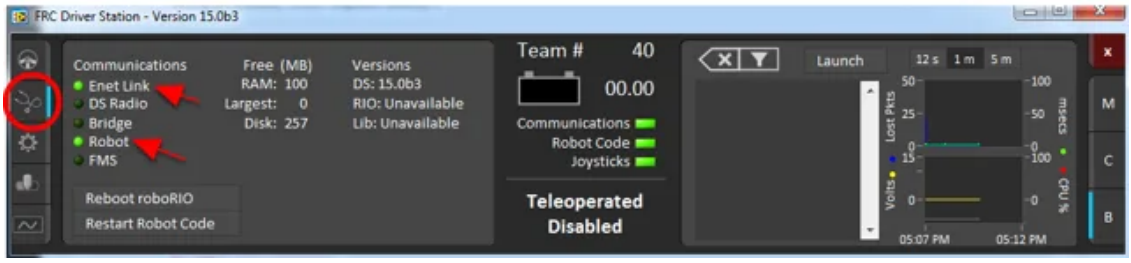


O DS deve ser configurado com o número da sua equipe para se conectar ao seu robô. Para fazer isso, clique na aba Configuração e insira o número da sua equipe na caixa de número da equipe. Pressione Enter ou clique fora da caixa para que a configuração tenha efeito.

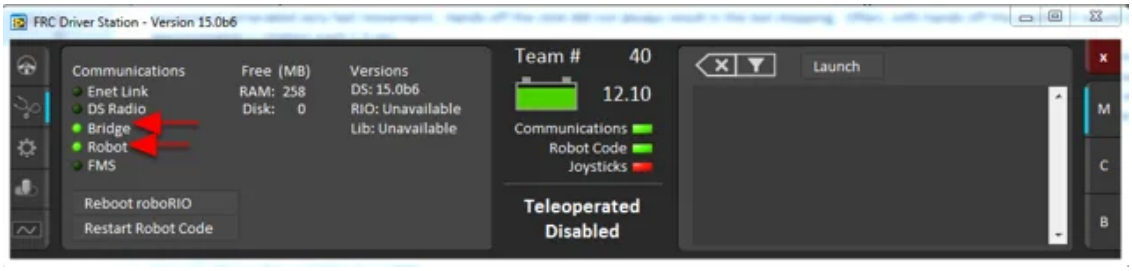
Normalmente, os PCs já têm as configurações de rede corretas para que o DS se conecte ao robô, mas, se não, certifique-se de que seu adaptador de rede esteja configurado para DHCP.

Confirmação de conectividade

Cabo

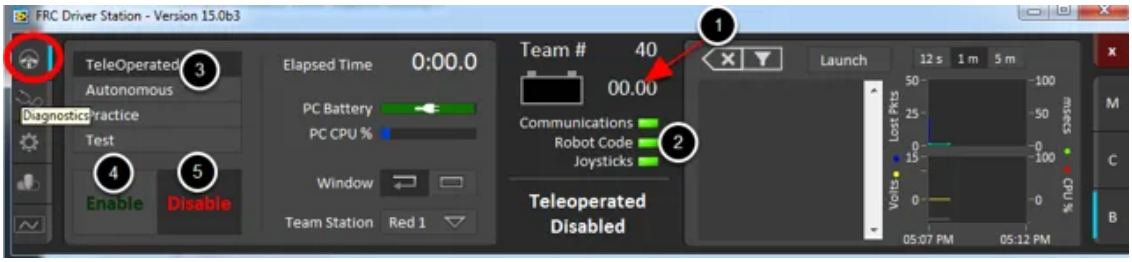


Sem fio



Usando o software da Estação do Condutor, clique em Diagnóstico e confirme que as luzes de Enet Link (ou luz do Rádio do Robô, se estiver operando sem fio) e as luzes do Robô estejam verdes.

Operando o robô



Clique na aba Operação

1. Confirme que a voltagem da bateria é exibida
2. Os indicadores de Comunicação, Código do Robô e Joysticks estão verdes.
3. Coloque o robô no Modo Teleop
4. Clique em Habilitar. Mova os joysticks e observe como o robô responde.
5. Clique em Desabilitar.

Operação sem fio

Antes de tentar a operação sem fio, a operação com cabo deve ter sido confirmada conforme descrito em Operação com Cabo. Executar o seu programa de teste enquanto conectado à Estação

do Condutor via WiFi confirmará se o ponto de acesso está configurado corretamente.

Configurando o ponto de acesso

Consulte o artigo "Programando seu rádio" para obter detalhes sobre a configuração do rádio do robô para uso como ponto de acesso.

Após configurar o ponto de acesso, conecte a estação do condutor ao robô sem fio. O SSID será o número da sua equipe (conforme inserido no Utilitário de Configuração da Ponte). Se você definiu uma chave ao usar o Utilitário de Configuração da Ponte, precisará inseri-la para se conectar à rede. Certifique-se de que o adaptador de rede do computador esteja configurado para DHCP («Obter um endereço IP automaticamente»).

Agora você pode confirmar a operação sem fio usando as mesmas etapas em "Confirmar Conectividade e Operar o Robô" acima.