

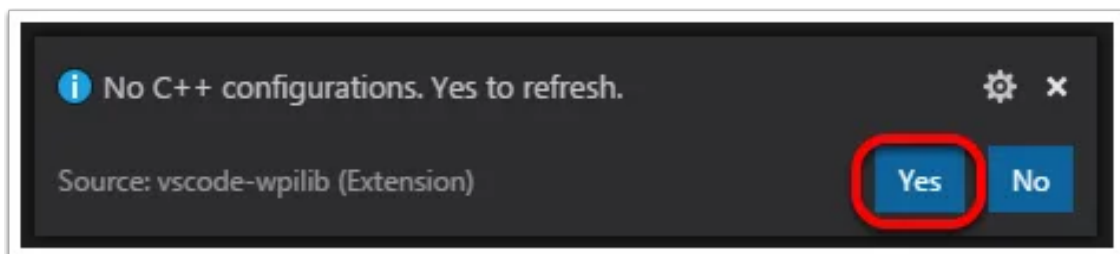
# Programando seu robô (C++)

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.

---

Revisão #1

Criado 19 dezembro 2023 18:21:56 por Enzo Coutinho

Atualizado 19 dezembro 2023 18:32:07 por Enzo Coutinho