Programming a Robot Using C++

Philipp Schrader & Tom Brown
October 27, 2012
The robot has mechanical systems and electrical hardware, but needs a program to tell it what to do

The program collects inputs from the drivers and sensors, and uses them to decide what motor output should be

Different programming “languages”
LabVIEW, C++, Java
C++ Overview

Invented in the 1980’s

Built as an extension on top of C

Object-oriented programming language
Why C++ in FRC?

Powerful and fast language; used widely in industry

Steep learning curve, but after that development is fast

Programming tools are less complicated, smaller, faster than LabVIEW
What is programming?

Writing instructions for machines to perform a set of desired tasks

It's about controlling how inputs to a system affect the outputs of the system.
Source Control

Centralizing where your source code is stored.

Acts as a backup storage, history browser, and versioning for the code.

Easily shows differences between two versions of the code. This includes the "current working version" with any modifications.
Wind River

The Windows program used to compile code into programs for the robot.

Primarily serves as a development environment for writing and debugging code.
WPILib

Already-written code provided by FIRST to make robot programming easier

Consists of classes that represent all common robot hardware

Example: Compressor, DigitalInput, Victor, DriverStation, Solenoid, Gyro
Simple Robot

SimpleRobot is a starting template to help teams quickly get a robot running.

There are other templates such as IterativeRobot, but we will focus on SimpleRobot.
```cpp
#include "WPILib.h"

class RobotDemo : public SimpleRobot
{
public:
    RobotDemo(void) {}
    void Autonomous(void) {}
    void OperatorControl(void) {}
};

START_ROBOT_CLASS(RobotDemo);
```
Simple Robot Cont'd

RobotDemo() is the constructor. This is where you initialize all members of the class.

Autonomous() is the function that gets called during the autonomous or hybrid period of a match.

OperatorControl() is the function that gets called during teleop. Function gets called once and needs a loop to continue operator control.
Drive with Joystick

// Declare drive components
Joystick *stick;
Victor *leftDrive;
Victor *rightDrive;

// Initialize components with port numbers
RobotDemo(void) {
    stick = new Joystick(1);
    leftDrive = new Victor(1);
    rightDrive = new Victor(2);
}

Philipp Schrader & Tom Brown - October 27, 2012
void OperatorControl(void) {
    while (IsOperatorControl()) {
        float leftY = stick->GetRawAxis(2);
        float rightY = stick->GetRawAxis(4);

        // Tank drive
        leftDrive->SetSpeed(-leftY);
        rightDrive->SetSpeed(rightY);

        Wait(0.005);
    }
}
Sensors

Sensors help the operators with controlling the robot.

They are crucial in autonomous, but also very valuable for teleop.

For example, appropriate sensors can be used to precisely position an arm on the robot.
AnalogChannel *plowPot;

RobotDemo(void) {
    plowPot = new AnalogChannel(2);
}

void OperatorControl(void) {
    while (IsOperatorControl()) {
        printf("%d", plowPot->GetValue());
        Wait(0.005);
    }
}
In our example, we are getting a feeling for how the potentiometer behaves.

Some sensors have their own classes in WPILib. Use them whenever possible.

Base Classes:
AnalogChannel, DigitalInput

Examples:
Encoder, Gyro, UltraSonic
Simple Feedback

We can use sensors to perform more sophisticated maneuvers than a human alone could do.

For example, we can move an arm into the best scoring position consistently on the push of a button.
Simple Feedback Cont'd

```
Relay *leftPlow;
Relay *rightPlow;

RobotDemo(void) {
    leftPlow = new Relay(1);
    rightPlow = new Relay(2);
}

void SetPlow(Relay::Value direction) {
    leftPlow->Set(direction);
    rightPlow->Set(direction);
}
```

Philipp Schrader & Tom Brown - October 27, 2012
void OperatorControl() {
    while (IsOperatorControl()) {
        if (stick->GetRawButton(2)) {
            SetPlow(Relay::kForward);
        }
        else if (stick->GetRawButton(3)) {
            SetPlow(Relay::kReverse);
        }
        else {
            SetPlow(Relay::kOff);
        }
    }
}
The previous slide shows a simple plow control without sensors. Button 2 lowers the plow. Button 3 raises the plow.

If we add sensor feedback to the code, we can prevent the plow from going past mechanical limits.

The next slide will incorporate feedback to the plow to prevent breaking itself.
/ Prevents the plow from going too far
if (stick->GetRawButton(2) &&
    (plowPot->GetValue() > 60)) {
    SetPlow(Relay::kForward);
}
else if (stick->GetRawButton(3) &&
    (plowPot->GetValue() < 250)) {
    SetPlow(Relay::kReverse);
}
else {
    SetPlow(Relay::kOff);
}
According to the pneumatics setup and electrical rules by FIRST, pneumatics have special programming requirements.

Solenoids and compressors have their own class in WPILib.

Even if the compressor is not on the robot, it needs to be controlled by the robot itself.
Pneumatics Cont'd

```c++
Solenoid *openGate;
Solenoid *closeGate;
Compressor *compressor;

RobotDemo(void) {
    openGate = new Solenoid(1);
    closeGate = new Solenoid(2);
    compressor = new Compressor(9, 5);
}
```
void OperatorControl(void) {
    openGate->Set(false);
    closeGate->Set(true);

    while (IsOperatorControl()) {
        if (stick->GetRawButton(1)) {
            openGate->Set(true);
            closeGate->Set(false);
        } else if (stick->GetRawButton(4)) {
            openGate->Set(false);
            closeGate->Set(true);
        }
    }

...
Pneumatics Cont'd

...  

    // Only runs compressor until 120 psi  
    if (!compressor->GetPressureSwitchValue()) {  
        compressor->Start();  
    }  
    else {  
        compressor->Stop();  
    }  

    Wait(0.005);  

}  

Philipp Schrader & Tom Brown - October 27, 2012
The autonomous code works similarly to the teleop code.

```c++
void Autonomous(void) {
  while (IsAutonomous()) {
    // Your code here
    ...
  }
}
```
For this example we will make the robot lower and raise its plow until the end of the autonomous period.

We will make use of a simple state machine that cycles between a state that raises the plow and a state that lowers the plow. We shall modify our SetPlow function as well.

Switching between the states is done through the potentiometer.
bool SetPlow(Relay::Value direction) {
    if ((direction == Relay::kForward &&
        plowPot->GetValue() > 60) ||
        (direction == Relay::kReverse &&
        plowPot->GetValue() < 260)) {
        leftPlow->Set(direction);
        rightPlow->Set(direction);
        return false;
    }
    leftPlow->Set(Relay::kOff);
    rightPlow->Set(Relay::kOff);
    return true;
}
void Autonomous(void) {
  int state = 1;

  while (IsAutonomous()) {
    switch (state) {
      case 1:
        if (SetPlow(Relay::kForward)) {
          state = 2;
        }
        break;
      ...
    }
    break;
  }
}
case 2:
    if (SetPlow(Relay::kReverse)) {
        case = 1;
    }
    break;
}
Philipp Schrader  
Firmware Developer, Nuvation Research  
phil@spartonics.org

Tom Brown  
Senior Software Developer, F.A. Servers  
tom.brown@spartonics.org