Getting Started with FRC Programming

GeorgiaFIRST Symposium
November 5, 2016

Gary Lewis
Mentor, Kell Robotics, Team 1311

Presenter Background

- Mentor, Kell Robotics, Team 1311
- Professor Emeritus of Computer Science and Physics, Kennesaw State University
- Ph.D., Physics, Georgia Tech
- Industrial experience
- Experience in electronics, chip manufacturing, embedded systems, robotics ...

This Presentation

- Oriented toward people new to FRC programming.
- Background information suitable for programmers and non-programmers.
- A copy of this presentation is available at the Kell Robotics GitHub site:

    http://github.com/kellrobotics/

- Look under "FRC-Programming"
- Additional supplementary material will be posted in the next few days.

A Very Important Page

- The FRC documentation is essential. You should step though it in setting up your robot, driver station, and programming station.
- Then, go back to it when you have issues.
- Currently at:

  https://wpilib.screenstepslive.com/s/4485
- A search on “FRC Control System” should also get you there.

FRC Programming Languages

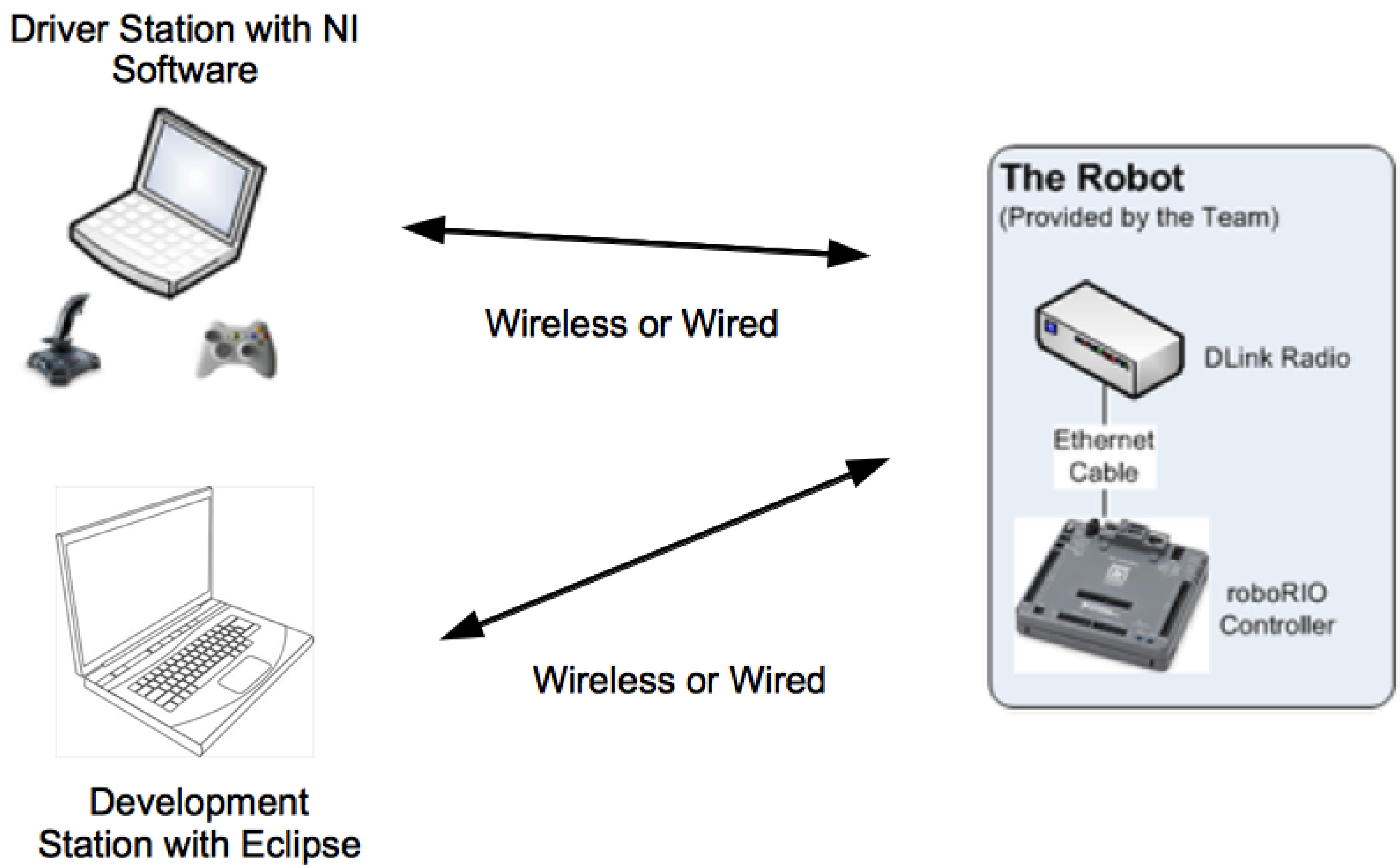
- The official languages that you can use to program an FRC competition robot:
  - Labview (graphical programming environment)
  - Java
  - C++
- I will use some Java examples. The FRC C++ libraries have the same functions as the Java libraries, so my discussion will largely apply to that as well.

Getting Started in Programming

- For our team members who have little or no background in programming, I have a document called “Jumpstart for FRC Java Programming”.
- That, along with some sample code, will be posted soon at the Kell GitHub site, at the location mentioned previously.
- It starts with some simple, not-robot programs, and works up to a very basic robot program.
- We expect programming team members to work with an experienced member and also work through a more extensive online programming course.

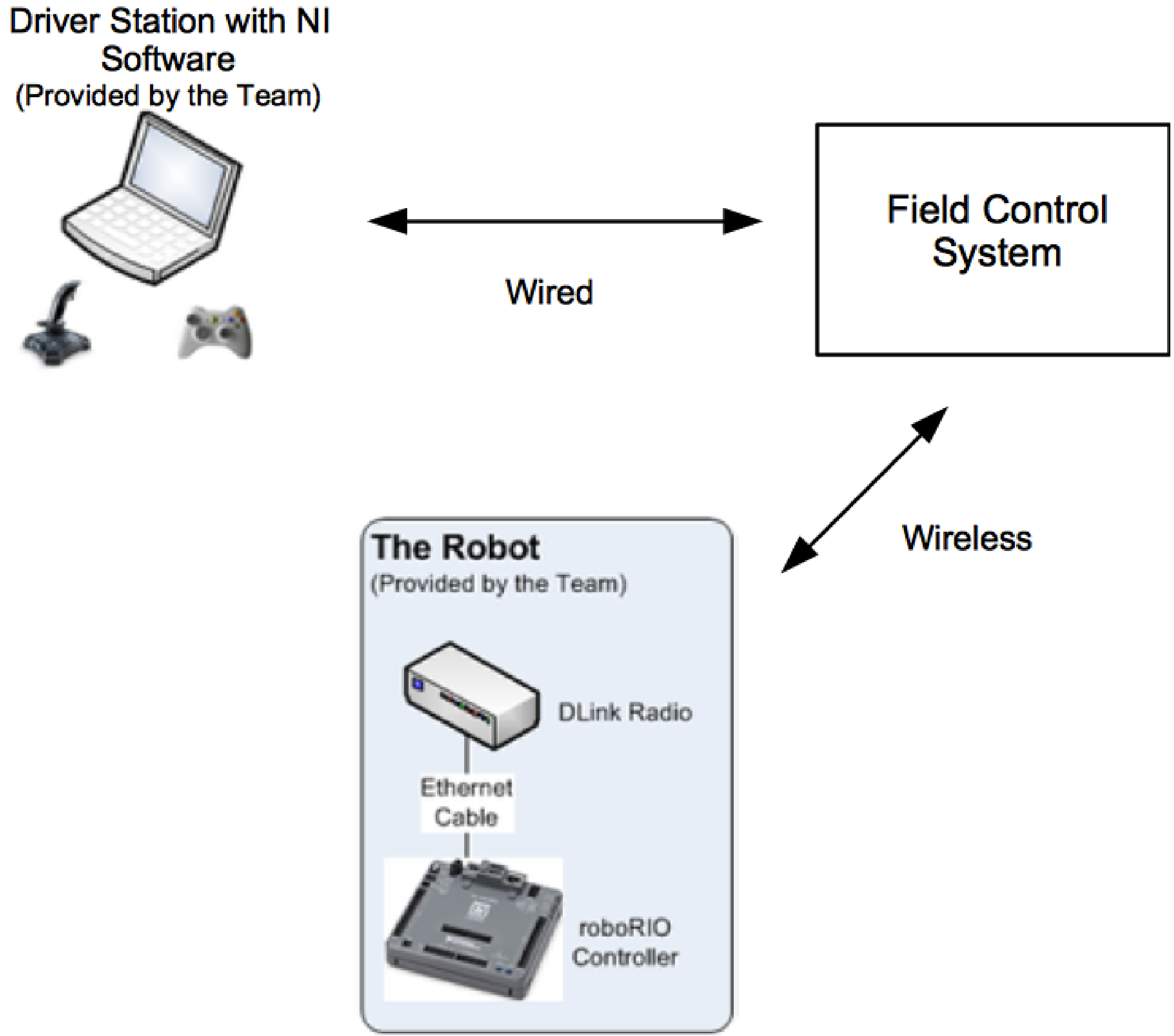
A Different Kind of Programming

- Some things that make FRC programs different from a typical program in an introductory course:
  - They are highly event-based, with input from different user input devices **and** a variety of sensors.
  - Three systems are involved: robot, development, and driver station.
  - Because your program must be controlled by the field system during competition, it has to run under a very specific framework provided by FRC. This is accomplished by having your program consist of classes that extend classes from the FRC library.



Development Setup

- Development and Driver Station can be on same computer.
- However, note the lack of connection between Development and Driver Station.



Competition Setup

Background Info

- FRC adopted a new control system n 2015 – roboRIO, by National Instruments
  - roboRIO uses an ARM processor running Linux, but this is effectively hidden from the programmer and users.
- The previous system, the cRIO, was used from 2009-2014, so the RoboRIO should be around for several years.
- The roboRIO, like the previous system, can be programmed in LabView, Java, or C++

Development System

- The roboRIO, like the previous system, can be programmed in LabView, Java, or C++
- Both Java and C++ use the Eclipse programming environment for the roboRIO
- The Eclipse development system can be installed and used on Windows, Mac, or Linux.
- Driver station and imaging software runs only on Windows.

Special Note

- If you are using Eclipse for some other purpose, **do not** try to add the FRC capabilities to that existing configuration.
  - There are too many opportunities for version and library incompatibilities.
- Do a new, parallel install of the FRC system with a separate copy of Eclipse – this works quite well, as long as you make sure the workspaces are separate.
- However, it is quite possible to install both the Java and C++ capabilities for FRC on a single Eclipse install.

Initial Setup
(See FRC Docs for Details)

- Download and install NI software to driver station.
- Firmware update to roboRIO, if needed.
- Image roboRIO, if needed (loads Linux system)
- Install Java Runtime on roboRIO (must be redone anytime above items are redone).
- Download and install Eclipse to development station(s). (Must be version specified by FRC.)
- Add FRC plugins to Eclipse.

Networking Notes

- The “standard” development setup is to configure the “Robot Radio” (wifi router) as an access point, so that it will appear as a local-only wifi network with the team number (ex. 1311).
- Both the driver station and the development station connect to this network, with IP addressed getting configured by a combination of DHCP and mDNS. (When things go well.)
- MDNS = multicast DNS service discovery

Networking Notes II

- On Windows, it may be necessary to load the NI software before mDNS will work properly.
- When directly connected by USB to the roboRIO (for initialization and upgrading), the roboRIO address will be 172.22.11.2
- More...

Networking Notes III

- IP addresses will normally contain the team number, indicated here by TEAM.
- The “Robot Radio” (router) address will be: 10.TE.AM.1 (10.13.11.1 for Kell Robotics)
- The roboRIO will normally be 10.TE.AM.2
- Driver Station and Development will have assigned addresses like 10.TE.AM.x, if the software has the team number set.
- Multiple Development Stations can connect, but only one Driver Station.

Major System Components

- RoboRIO
- Power Distribution Panel
- Pneumatics Control Module
- Voltage Regulator Module
- Motor Controllers
- Specified WiFi Router (“Robot Radio”)

Starting a Programming Project

- From within Eclipse-
  - File → New → Project...
  - WPILib Robot Java Development → Robot Java Development
- You will then need to choose the base type for the project:
  - Sample
  - Iterative
  - Command-Based

Sample Base Type

- Formerly “Simple Robot” - probably changed because people were assuming this should be the default choice.
- It is not recommended except:
  - For simple testing, maybe, but really ... don’t do it.
  - As a base for very complex programming that does not fit into the other types, but only if you really know what you are doing.

Iterative Base Type

- Recommended starting place for most people
- Provides a framework where your code is called about every 20 ms.
- But, make sure you understand the purpose of the Command-Based type, so that you don't build a very complex Iterative design when you should move on to Command-Based.

Command-Based Base Type

- Not a true separate base type, really just an extension of Iterative.
- Provides a structure for task management that is very helpful for complex autonomous code.
- Takes advantage of object oriented features and encapsulation of code.

Simple Drive Program

- To build a program, choose Iterative as the base type.
- This will generate files with some existing structure. Your code will go into Robot.java.
- Robot.java contains class Robot that extends class IterativeRobot.
- IterativeRobot is a class from the FRC library that will give the underlying structure allowing your robot to operate as part of a competition field.

Simple Drive Program II

- Robot.java contains several methods that get called by the FRC system at appropriate times:
- robotInit() - called once when robot starts
- teleopInit() - called once when robo enters teleop mode
- teleopPeriodic() - called repeatedly (about 50 times per second) when robot is in teleop mode
- autonomousInit, autonomousPeriodic, testInit, testPeriodic are the corresponding routines for those modes.

Periodic Methods

- Some of the methods on the previous page may not be inserted automatically, but they will work.
- Most of the action of the program is done in periodic modes, which makes for a very different programming environment than many people are used to.
  - Have to think in terms of repeated execution of that code.
  - No loops or time delays – check it, do it, allow the program to continue.

Example TeleopPeriodic

```
public void teleopPeriodic() {
      double leftside = leftStick.getY();
      double rightside = rightStick.getY();
      if (Math.abs(leftside) > .2) {
         leftDrive.set(-leftside);
      }
      else {
         leftDrive.set(0);
      }
      if (Math.abs(rightside) > .2) {
         rightDrive.set(rightside);
      }
      else {
         rightDrive.set(0);
      }
}
```

Drive Program

- The previous code could be even more compact if you use the RobotDrive class, which handles things in a single class (with some reduction in flexibility).
- In addition to that previous code, the program will require some declarations and initialization.
- I will post a complete program in the next few days.

Deploying to Robot

- Once the program is entered into Eclipse,
  - Make sure the development system is connected to the robot WiFi.
  - Make sure the team number is correctly set in Eclipse.
- Right click on the project, then select
  - Run As...
  - WPILib Java Deploy

Program Load

- The development system will attempt to connect and load.
- Messages will appear in the console window in the lower right.
- Be aware that some things that look like errors in that window are not – for example you can get a message because the system failed to kill a previous program because the robot was just started.

Interaction with the DS Console

- If you start up the driver station console and connect it to the robot with a running program, it should indicate a connection.
- You can then switch to teleop, or other, mode, and begin testing.
- If the robot is on, the program will continue to run until the robot is restarted.

Modes

- Autonomous and Teleop modes are where you put code to execute during those times in competition.
- When not in competition, you can switch between them with the driver station.
- Test mode enables some special features in terms of displaying information. It is not required for competition.

Debugging

- Riolog console
- Debugger in Eclipse
- Displaying info on Driver Station (variables, etc.)

Debugging - Riolog

- Enable the plugin in Eclipse:
  - Window → Show View → Other...
  - General → Riolog
- This will add a “Riolog” tab to the lower right window in Eclipse
  - That window displays a few error messages from the roboRIO.
  - It also displays anything written to console by Java. Example: System.out.println(“Here I am”);

Note on Consoles

- “Console” in Eclipse window
  - Shows Linux console
- “Riolog” in Eclipse window
  - Shows console output from Java program

Sensors and Encoders

- A variety of sensors and encoders are usable on the robot.
- One of the first that is normally used is a rotational encoder called a quadrature encoder. This allows measuring relative distances traveled when connected to the drive train.
- The quadrature encoder requires two digital I/O ports because it sends two digital signals. The library routines allow specifying two ports for these.

More Sensors

- A second common sensor is a gyro for measuring angle.
- The most common simple gyro connects to an analog port on the roboRIO.
- In fact, it is limited to analog port 0 or 1 because it requires a hardware accumulator in the processor.

Motors

- Motors and other motion devices are very carefully limited by FRC.
- Motor controllers are connected to the robotRIO either via PWM ports or the CAN bus.

Compressed Air Systems

- The Pneumatic Control Module (PCM) controls both the compressor and air solenoids.
- The PCM is controlled by the CAN bus.
- Solenoids are either on or off – no proportional control is possible with allowed equipment.
- The PCM is not needed is compressed air is not used.

When to Move to Command-Based

- If you are planning autonomous mode operation that is more complex than “drive in a straight line”
- If you are using more than a couple of buttons to control action on the robot.
- If you are using encoders and other feedback devices in more than a very basic mode.

PID Systems

- Needed in feedback systems to allow approaching a point smoothly
- Proportional, Integral, Derivative

Questions?