Noteworthy Features of the Average Joes' 2018 FRC Code

Doug Wegscheid

January 3, 2019

Abstract

FRC Team 3620, "The Average Joes", from St. Joseph High School in St. Joseph, Michigan, is releasing it's 2019 Java code for other FRC teams to learn from. The code is available on github: `https://github.com/FRC3620/FRC3620_2018_CoffeePi.git`

This paper is a summary of what the team programming mentor considers to be features that other FRC teams may want to look at and use: creation of autonomous CommandGroups on-the-fly (based on game data), use of PathFinder with inheritance, examination of the CAN bus, event logging, and data logging.

Some of the information contained here was not discovered or developed by The Average Joes; much of it was gleaned from other team's postings of code, discussions on Chief Delphi, or other publicly available sources. We didn't generally keep track of where we got ideas that we reused, and apologize for not being able to attribute them properly.

1 Overview

Our 2018 FRC programming was done in Java command-based code, using RobotBuilder for the initial setup. We then abandoned RobotBuilder when adding more hardware (keeping the RobotBuilder project and our code in sync has not worked well in previous years). We did version control with git, using github as our central repository.

There are 4 features of the 2018 code that may be of use to other teams:

- Creation of autonomous CommandGroups on the fly

  Our autonomous had the ability to place the first cube in the switch or on either end of the scale.

  This was a two-stage process. When the game specific message came available, code in AutonomousDescriptorMaker would lookup the correct path to use and at what height to drop the first cube based on our initial position, the game specific message, and whether or not we trusted the center robot to reliably place a cube in the switch. This initial path and desired drop height were placed in an AutonomousDescriptor.

  The AutonomousDescriptor was used to build a CommandGroup on the fly, with the correct commands to navigate to the first cube drop, get our elevator to the correct height, and drop the cube. Additionally, if we were dropping in the scale, additional commands were added to acquire another cube via dead reckoning[1] and drop it[2].

  In prior year's testing, we found that it was not possible to put the same command in a CommandGroup in successive test runs (this is enforced in the WPIlib Command.setParent()). As a hack, the code for creating the command groups on the fly actually looks up the class of the passed in commands, then creates new instances of those classes. This would not work well for commands that took variables in their constructors.

- Use of Pathfinder with inheritance

[1] "blind sow acquisition"

[2] If you look at this code and are suspicious of usingthe Path2_AlleyCube_LeftScaleSide command regardless of which side of the scale we are working, so am I.

We moved away from our previous set of autonomous commands to move the robot in a straight line or turn a certain amount to using Pathfinder to navigate during autonomous to using Pathfinder. It took a great deal of PID tuning to get Pathfinder functioning well; lack of access to the PID error made tuning difficult.

We ended up with over 30 paths for our autonomous. Rather than duplicate the code for following a path 30 times (with the resultant problem of having to update it in 30 places as we refined it), we put that code into an abstract AbstractPath class. That class had an abstract method to provide the waypoints, as well as overridable methods to provide various PID tuning parameters. The class for the individual paths only needed to provide the waypoints, along with any PID parameters that needed changing.

This was all in package `org.usfirst.frc.team3620.robot.paths`.

- Examination of the CAN bus

  We have several test boards with RoboRIOs on them, not all the test boards are configured identically. If code is deployed to a RoboRIO that instantiates WPIlib objects for CAN bus devices, and the devices are not present on the CAN bus, then the console will get flooded with error messages. This makes testing difficulty.

  We carried over code from 2017 to 2018 to examine the CAN bus, see what devices are present, and only instantiate the corresponding WPIlib object if the corresponding physical device is present[3]. Code using the WPIlib objects would have to guard for null values in case the CAN device was not present (and thus the WPIlib object was not instantiated).

- Event Logging

  Event logging is the timestamped logging to a file of significant events during the execution of robot code. Events worth logging could include:

  - transitions between autonomous, teleop, test, and disabled states.
  - initialization, end, and interruption of commands.
  - blockage of operation because of limit switches.
  - acquisition and loss of vision targets.
  - brownouts.
  - selection of different autonomous modes.

  This code was largely carried over from 2017, some of the code needed to be changed to mesh with the changes to the DriverStation API.

  Code is in package `org.usfirst.frc3620.logger`.

  Figure 1 is an example of an event log.

- data logging

  Data logging is the continuous timestamped logging of significant sensor data, actuator status, power statistics (voltage and current draw), and operator inputs to a file. This data could be processed after a test session, practice session, or match using Excel, Veusz, gnuplot, or some other graphics or analytical software to provide plots of such data. Figure 2 is an example of such a data log.

  This code was not changed from 2017 to 2018.

  Code is in package `org.usfirst.frc3620.logger`.

  In past years, this data was used to determine if all motors in a multi-motor drive system were working, and used to determine the duty cycle of the air compressor.

[3] we actually cheated; for subsystems with multiple motor controllers, we only checked for the first controller.

```
2018/04/28 11:59:47.183 [org.usfirst.frc.team3620.robot.Robot] INFO - CAN bus: [PDP 0, PCM 0, SRX 1, SRX 5, SRX 9, SRX 13, SRX 14, SRX 15, SPX 3, SPX
       4, SPX 7, SPX 8, SPX 10, SPX 11, SPX 12]
2018/04/28 11:59:47.796 [org.usfirst.frc.team3620.robot.subsystems.DriveSubsystem] INFO - Resetting X Displacement, X = 0.0
2018/04/28 11:59:47.805 [org.usfirst.frc.team3620.robot.subsystems.DriveSubsystem] INFO - Resetting NavX Angle, Angle = 0.0
2018/04/28 11:59:47.827 [org.usfirst.frc.team3620.robot.subsystems.IntakeSubsystem] INFO - Initializing intake subsystem
2018/04/28 11:59:47.885 [org.usfirst.frc.team3620.robot.OperatorView] INFO - /dev/video0 exists, starting the camera server
2018/04/28 11:59:48.579 [org.usfirst.frc.team3620.robot.ControlPanelWatcher] INFO - starting control panel watcher
2018/04/28 11:59:48.589 [org.usfirst.frc3620.misc.PersistentChooser] INFO - loading chooser trustChooser from preferences: value is Yes
2018/04/28 11:59:48.600 [org.usfirst.frc3620.misc.PersistentChooser] INFO - loading chooser posChooser from preferences: value is Left
2018/04/28 11:59:48.611 [org.usfirst.frc3620.misc.PersistentChooser] INFO - cannot find chooser delayChooser in preferences: trustChooser=Yes .type=
      RobotPreferences posChooser=Left
2018/04/28 11:59:48.620 [org.usfirst.frc.team3620.robot.ControlPanelWatcher] INFO - Control Panel appears to be a vJoy Device
2018/04/28 11:59:48.726 [org.usfirst.frc3620.logger.DataLogger] INFO - setupOutputFile filename is 20180428-115947.csv
2018/04/28 11:59:48.729 [org.usfirst.frc3620.logger.DataLogger] INFO - setting file to /home/lvuser/logs/20180428-115947.csv
2018/04/28 11:59:48.739 [org.usfirst.frc3620.logger.DataLogger] INFO - DataLogger interval = 1000ms
2018/04/28 11:59:48.748 [org.usfirst.frc3620.logger.DataLogger] INFO - setupOutputFile filename is 20180428-115947.csv
2018/04/28 11:59:48.751 [org.usfirst.frc3620.logger.DataLogger] INFO - setting file to /home/lvuser/logs/20180428-115947.csv
2018/04/28 11:59:48.766 [org.usfirst.frc.team3620.robot.Robot] INFO - Switching from INIT to DISABLED
2018/04/28 11:59:48.778 [org.usfirst.frc.team3620.robot.RobotMap] SEVERE - Device SPX 2 cannot be found on the CAN bus
2018/04/28 11:59:48.764 [org.usfirst.frc3620.logger.DataLogger] INFO - Writing dataLogger to /home/lvuser/logs/20180428-115947.csv
2018/04/28 11:59:48.783 [org.usfirst.frc.team3620.robot.RobotMap] SEVERE - Device SPX 6 cannot be found on the CAN bus
2018/04/28 11:59:48.861 [org.usfirst.frc.team3620.robot.subsystems.IntakeSubsystem] INFO - Pivot home, reseting encoder
2018/04/28 11:59:50.074 [org.usfirst.frc.team3620.robot.Robot] INFO - Lift set to high gear
2018/04/28 11:59:50.081 [org.usfirst.frc.team3620.robot.Robot] INFO - Clamper closed
2018/04/28 11:59:50.084 [org.usfirst.frc.team3620.robot.Robot] INFO - Switching from DISABLED to TELEOP
```

Figure 1: Event log sample

```
time,timeSinceStart,robotMode,robotModeInt,batteryVoltage,drive.l1.power,drive.l1.voltage,drive.l1.current,drive.l3.voltage,drive.l4.voltage,drive.r1
      .power,drive.r1.voltage,drive.r1.current,drive.r3.voltage,drive.r4.voltage
04-28-2018 11:59:48.979,0.241133,DISABLED,1,13.13,0,0,0.12,0,0,0,0,0.12,0,0
04-28-2018 11:59:49.759,1.020973,DISABLED,1,13.12,0,0,0.12,0,0,0,0,0.12,0,0
04-28-2018 11:59:50.759,2.020621,TELEOP,3,13.01,-0.01,0,0.12,0,0,-0.02,0,0.12,0,0
...
```

Figure 2: Data log sample

2 Event Logging

We used the JRE provided java.util.logging framework to do the actual logging[4], but used the SLF4J logging API as our mechanism for getting log events into java.util.logging. If the runtime on the roboRIO becomes complete enough to support log4j2, we would be able to use that (with it's configuration and performance advantages over j.u.l), and keep the same logging statements that we have always used.

We have a EventLogging.commandMessage (org.slf4j.Logger logger) method to be called from Command.init(), Command.end(), and Command.interrupted(). This simply logs the command and the method name. It's the same as logger.info("...") methods, but does not need to be updated as code is refactored or copied.

[4]We originally considered using Apache log4j 2 instead of java.util.logging, but log4j has dependencies on parts of the Java runtime that were not present in the compact version of the Java runtime that FIRST has us install on the roboRIO in past years. We may look at this again in 2019.