Programming 101

Introduction

This guide was initially created by The Compass Alliance and revised by *FIRST*®. The Compass Alliance was founded by 10 teams from around the world with the mission of helping *FIRST*® Robotics Competition teams sustain and grow.

This resource will cover the basics of programming in *FIRST* Robotics Competition. It covers C++, Java/Kotlin, LabVIEW, and Python.

Getting Your Robot Running

Picking a Programming Language: Java, C++, or LabVIEW

- **Java** is a textual language that is commonly taught at high schools and used for the AP CS exams. It is a "safe" language in that runs in its own virtual environment. While it doesn't generally affect *FIRST* Robotics Competition, this virtual environment, also known as the JVM, means that Java programs are noticeably slower than compiled languages when used for computationally intensive tasks. Java is often selected because of its ease of use, and cross-compatibility. Teams that use Java include 254, 125, 503, 4911, and 1241.
- **C++** is a fast textual programming language. It is used in industry for real time systems because of its power and efficiency, but the learning curve is much steeper than Java. C++ evolved from the C programming language and the mixture of historical and modern features sometimes lead to confusing syntax and/or unexpected behaviors. *FIRST* Robotics Competition teams primarily use it due to its speed, flexibility, and its extensive mathematical libraries. Teams that use C++ include 1756 and 6800.
- **LabVIEW** is a graphical dataflow programming language developed by National Instruments (NI) for use by engineers and technicians. In a LabVIEW diagram, it is very easy to take advantage of advanced computing features, such as running pieces of code in parallel. While powerful, such features often introduce new issues to deal with. NI provides extensive debugging tools, however. The LabVIEW environment and language come with its own learning curve and unique challenges. *FIRST* Robotics Competition teams primarily use it due to its simplified graphical syntax and extensive engineering libraries. Teams that use LabVIEW include 2468 and 4150.
- **Python** is a high-level, general-purpose programming language that emphasizes code readability. Python is heavily featured in many online programming courses and is used frequently in academia. Unlike the other languages, Python is not compiled, making it far more likely for things like typos to lead to code crashes, so teams should make sure to use unit tests, simulation, and other recommend techniques to combat this. *FIRST* Robotics Competition teams primarily use it due to its ease of use and student and/or mentor familiarity. Teams that use Python include: 1736, 2881, and 4774

Choosing **what language always depends on what is easiest for your team**. For example, it can often make sense to choose a language because a programming mentor is an expert in that language. On the other hand, it might make sense to choose based on the ease of learning a given language. No matter what your decision, remember that the choice of programming language is specific to the

working environment and people of your team. All of the languages are capable, well supported, and sufficiently powerful for *FIRST* Robotics Competition use. If your team has no students or mentors with experience in any language and no direct mentoring from another team with a preferred language, we currently recommend selecting Java. This language is used by the vast majority of teams, meaning code examples are more readily available.

Teaching the Programming Language

When teaching programming for *FIRST* Robotics Competition, there are two distinct subjects that need to be taught. The first is the semantics and syntax of the programming language itself, and the second is interfacing with *FIRST* Robotics Competition components. A guide to learning each language can be found below:

- C++ – Learn CPP
- Java – Codecademy
- LabVIEW – LabVIEW Courses
- Python – List of Python Guides

Picking your starting code

- For **Java and C++**, there are four different "classes" that can be used when interfacing with the Robot. A comparison of these classes, and what they mean can be found on the Creating a Robot Program page.
- For **Python**, there are two different recommended frameworks that you can learn more about on the Frameworks page of the RobotPy documentation.
- For **LabVIEW**, the starting templates include a mixture of timed, iterative, and spawn/abort mechanisms. The templates are described in this Robot Framework Tutorial.

Once your team has picked a programming language and has started coding, the *FIRST* Robotics Competition Docs site is a good resource on how to set up your development environment and get code onto your robot. These guides are invaluable for *FIRST* Robotics Competition programming:

- Zero-to-robot including wiring and software installation
- C++\Java
- LabVIEW
- Python

Getting code onto your robot!

- C++/Java
  - C++ and Java code must first be "built" and then deployed onto the robot to run. Learn more about this process in the Building and Deploying Robot Code article.
- LabVIEW
  - For development, you should run from source.
  - Once complete, you will deploy a built executable.
- Python
  - Python code isn't "built" like the other 3 languages, it is "interpreted" at runtime, but it still must be deployed to the robot. To learn how to do that, see the Deploy Python program to roboRIO article.

Code for mechanisms

- For C++, Java, and Python, simple code for controlling a tank drive can be found in Using the WPILib Classes to Drive your Robot.
- For LabVIEW, the simple drive code is in the template in the TeleOp VI.
- Most *FIRST* Robotics Competition robots have actuated mechanisms other than the drivetrain. This could be anything from a spinning flywheel to a pneumatic catapult. All these mechanisms should be controllable in autonomous, or in teleop. To control mechanisms using speed controllers over PWM, there is a guide for C++, Java and Python in the Using Motor Controllers article. For LabVIEW, a guide can be found in Add an Independent Motor to a Project.
- If using speed controllers over CAN, you must utilize the library provided by the vendor in order to interact with them. For help installing these libraries, refer to the 3rd Party Libraries article from WPILib. To learn more about using each of these libraries visit their documentation:
    - CTR Electronics
    - REV Robotics
    - The ThriftyBot
    - Redux Robotics
- Teams with Swerve drivetrains will need some more complex code to control the drivetrain. While some teams still code this from scratch, or from constituent pieces provided by WPILib, many use one of the available libraries, examples, or tools:
    - Yet Another Generic Swerve Library (YAGSL) – supports nearly all hardware
    - CTR Electonics Swerve Project Generator
    - REV MaxSwerve Example
    - Thrifty Swerve Example (generator coming soon)
    - AdvantageKit Swerve Templates – options for both Spark and TalonFX (i.e. Kraken) motor controllers

Autonomous

C++/Java

- A video on creating very basic autonomous modes for TimedRobot programs can be found in the first video of the 0 to Autonomous series (note that videos 5 and 6 in this series contain outdated code).
- Autonomous routines in command programming are just command compositions. You can learn more about composing commands in the Command Compositions article.
- Many teams develop multiple autonomous routines to accomplish different game actions, and/or start at different locations on the field. For more information on an easy way to select between multiple autonomous routines, see the Choosing an Autonomous Program article. The images in this article show the SmartDashboard tool, but the robot code shown will also work with newer, more popular dashboards like Elastic.

LabVIEW

- The LabVIEW templates include autonomous code for jiggling the robot in place.
- You can modify motor power values and timing to accomplish many tasks as shown in 2468's autonomous code from 2017.

Level Two: Version Control and Advanced Motor Control

Version Control

- A “version control system” or VCS is a way of keeping track of saving your code that tracks changes made over time and allows you to go back to previous versions if needed. The most common VCS in current use is Git, with usage estimates of over 90%. Git can be used locally, to track versions on a single computer, or connected to a service such as GitHub to back up code to the cloud in case anything happens to your computer, or to share code between multiple computers. You can learn more about the basics of Git in the Git Introduction article from WPILib.
- Git can also be used directly from VSCode, you can learn more about that in this article from Microsoft about Git source control in VSCode.

PID Control

- PID Control is a type of closed-loop or feedback control which means it allows you to control a mechanism based on a desired target such as position, rather than voltage or % output. Using PID, you can tell an arm to turn to 30 degrees, instead of telling it to directly output a voltage. This is especially useful in autonomous. Being able to tell a robot to drive 5 meters instead of full power for 0.5 seconds allows for enhanced repeatability. PID uses a measured value (from a sensor) and compares to the target in order to determine the output to the motor.
- A video intro to PID control can be found in the WPILib documentation.

Feedforward Control

- Feedforward control is an open-loop technique to calculate the desired motor output to achieve a desired target such as a mechanism velocity. Feedforward utilizes information about the system, which can be either calculated or measured, to estimate the appropriate output to achieve the desired goal. This is different than PID control in that no information about the current state of the system is utilized in Feedforward control. The two types of control can also be combined to allow Feedforward to calculate the ideal response, while allowing PID to respond to disturbances.
- You can learn more about Feedforward in the WPILib Introduction to Feedforward.
- Then you can experiment with tuning controls on different types of common *FIRST* Robotics Competition mechanisms using the WPILib controls tutorials.

Motion Profiling

Another control technique commonly used in *FIRST* Robotics Competition is Motion Profiling. When utilizing PID and/or Feedforward control, quickly changing the desired target or “setpoint” may result in large instantaneous jumps in motor output. These abrupt changes may result in undesirable behavior such as wheels or belts slipping, or damage to gear teeth. To combat this issue, a motion profile can be used to smoothly adjust the setpoint in order to control acceleration (trapezoidal profile) or change in acceleration aka “jerk” (S-curve profile). This can be done using either WPILib or features built into some motor controllers when used over CAN:

- WPILib Trapezoidal Profiles
- CTR Electronics Motion Magic
- REV Robotics MAXMotion

Level Three: Vision and Path Planning

When you're ready to move beyond the basics, some common next topics include vision processing and path planning.

Vision Processing

There are two main uses of vision processing in *FIRST* Robotics Competition:

- **AprilTags** – AprilTags are specific visual markers that *FIRST* places in specific locations on the field to support two different types of uses. The first use is for direct targeting of areas of interest on the field such as sources of Scoring Elements, goals for Scoring Elements to be scored in/on, or structures for robots to interact with. The second use is for field localization, where a robot can utilize it's orientation and distance to one or more tags to calculate its location on the field. For more detail about how AprilTags work, check out the AprilTag Introduction from WPILib.
- **Object Recognition** – Object recognition can be used to detect any type of object on a *FIRST* Robotics Competition field, but the most common ones are Scoring Elements and other robots (generally via detecting Bumpers). This can be used for detecting Scoring Objects to acquire, determine the current scoring state (for games where Scoring Elements remain placed on a structure), or avoid other robots.

There a number of platforms teams use for performing vision processing:

- RoboRIO – The roboRIO is capable of performing vision processing, but is not powerful enough to achieve high frame rates, and may not be suitable at all depending on the demands of robot code. If you wish to just stream video back to your Driver Station computer, the roboRIO is typically capable of that alongside most teams robot code. You can learn more about how to do that in the Vision on the roboRIO section of the WPILib documentation.
- PhotonVision – A community developed, open source, vision application that runs on a secondary processor on the robot that provides support for camera streaming, AprilTag recognition, colored blob detection (can be used for simple object recognition), and machine learning powered object detection (on select platforms). Users must select and source their own co-processor, power supply and camera, recommendations can be found on the Selecting Hardware page of the PhotonVision documentation.
- LimeLight – LimeLight is a company that sells integrated smart cameras designed for *FIRST* teams. These cameras provide an integrated solution that contains the processor, power supply, and camera all in one. LimeLight software provides support for camera streaming, AprilTag detection, object recognition, or customized vision pipelines.

Path Planning

Path planning is the process of creating and following trajectories. This is generally used to create efficient, smooth robot movements between two desired locations, especially in autonomous mode. There are currently 3 main tool teams use for path planning:

- PathWeaver – This tool bundled as part of WPILib is a legacy tool that can be used for generating paths which can be followed using WPILib trajectory following library components. This tool has significant limitations when used with swerve drive (the robot heading cannot be decoupled from the path direction) so is typically only used by teams using differential drive (aka "tank drive") robots.

- PathPlanner – This community developed tool is currently the most popular tool for generating and following paths. By default, it is configured for swerve drives but can also be used with differential drive robots by disabling “Holonomic mode” in the app settings. The robot side library also contains functionality for “Pathfinding” or generating a path on-the-fly between two points while avoiding field obstacles. This is often used for automation during the teleop period when the robot may be starting from an arbitrary location and navigating to a specific point on the field. Pathfinding in teleop generally requires vision processing to help maintain the robot’s knowledge of its location on the field throughout the match.
- Choreo – Choreo is another community created tool to generate and follow mathematically optimal paths given a set of constraints (derived from information about the robot capabilities). Choreo is designed only around swerve drives and does not support differential drives.