Title page

Contents

1 Preface

Throughout this guide, we will be working to build a fully functional robot. At the end of this guide, section 10 is where I build a fully functional robot code that is based on the code written throughout this guide. The individual lines are not necessarily explained in that final section, but a walk-through of why certain files were created at what time, and a rough overview of what each file does can be found there. I highly encourage you to read this from start to finish and to not skip any sections even if you think you know what you are doing. Worst case scenario, you get to read some extra code ☺. I think it is really worth it. Plus, it's not like this guide is all that long. Also note that all the code can be found on my github.

2 Introduction

The first thing to know about FRC programming (at least when it comes to command based programming), is that every piece of your code will be split up into very distinctive sections. The most important of these sections are the Subsystems and the Commands. Each will be explained later, but for now just know that when you write code for your robot, one of the most important things to remember is to stay organized. Every class, every file, every function belongs in their own little section that should be easily identified.

3 An explanation of subsytems

Subsystems are a larger collection of motors and sensors all linked to doing one specific task on the robot. For example, all robots will have a drivetrain, some might have a claw, maybe an elevator, etc. These are all subsystems. On the drivetrain for example, you might have a set of motors which control the wheels. Then you might have something like an ultrasonic sensor to measure the distance to a wall, a gyro and a magnetic encoder to measure the wheels' displacement. This collection of motors and sensors are what we will call a subsystem. In code, we will need to make an abstraction for the motors and their controllers, and the various sensors on the subsystem. With these abstractions we can then control the motors using the information given off by the sensors. To use these motors, we put these in commands, which we

will cover in the next section. This is all to say that the subsystem is the actual thing that has the motors. We can define certain functionality that these subsystems might have (for example, an elevator might have an `elevate()` and a `fall()` function) which will then be used by commands. The subsystem is a tool that allows us to abstract this functionality to facilitate very complex behaviour that we want our robot to have.

4 An explanation of commands

Commands are the higher level structures that allow for the actual complex movement of the robot. They use the subsystems' information to calculate various outputs to feed into the subsystems' motor controllers. Commands are essentially actions that you make your robot take. When you want your robot to "raise the elevator", you would call the `raiseRlevator()` command. You can write simpler commands and then combine them together to get more intricate behaviour. For example, say you wanted to write a command to drop a cube that you have in your claw that is attached to you elevator. What you really want to do is first raise the elevator, then open your claw, then lower the elevator back down. Thus, you would need to write a `raiseElevator()` command, a `openClaw()` command and a `lowerElevator()` command. You would then combine all of these into another `dropCube()` command. We will see later on in section 8 how to do this sort of thing.

5 Basic code structure

The following is the basic structure that any command based pyFRC code should follow. This is the structure that this guide will follow.

```
FRC Robot Code
├── commands
│   └── A bunch of command files go here
├── subsystems
│   └── A bunch of subsystem files go here
├── sim
│   └── config.json
├── tests
│   └── pyfrcTest.py
├── physics.py
├── vision.py
├── oi.py
├── robotMap.py
└── robot.py
```

6 Implementing `robot.py`

`robot.py` is the main part of your robot. This is the file where it all begins. In here you define the very general behaviour of your robot for the following states the robot can be in:

- `autonomous`
- `teleop`
- `disabled`

Autonomous is the 15 seconds where your robot must follow preprogrammed instructions, teleop is when your robot is controlled by the driver and disabled is when your robot is doing nothing. Each one of these states has an `init()` and a `periodic()` function. The `init()` is when the robot first enters that state. For example, if you went from disabled to teleop, the state of the robot would evolve as follows:

$$\cdots \rightarrow \texttt{disabledPeriodic} \rightarrow \texttt{teleopInit} \rightarrow \texttt{teleopPeriodic}$$

These are all functions that we need to implement in our `robot.py`. The robot also has a `robotInit()` function that we need to worry about. So, putting this together, we need to write some sort of class that implements all of these functions. Here is a start (`code-samples/robot-py/robot-0.py`):

```
import wpilib

class Robot:
    '''
    The robot class
    '''
    def __init__(self):
        '''
        Do nothing for the constructor for now
        '''
        pass

    def robotInit(self):
        '''
        This will end up being used as the robot's
        constructor. This is the case for legacy reasons.
        '''
        pass

    def autonomousInit(self):
        '''
        This is what is called when the robot switches from
        any state to its autonomous state.
        '''
        pass

    def autonomousPeriodic(self):
        '''
        This function is called periodically during
        autonomous.
        '''
        pass
```

```
    def teleopInit(self):
        '''
        This function is called when to robot switches from
        any state (in practice only from the disabled state)
        to the teleop state.
        '''
        pass

    def teleopPeriodic(self):
        '''
        This function is called periodically during teleop.
        '''
        pass

    def disabledInit(self):
        '''
        This function is called
        '''
        pass

    def disabledPeriodic(self):
        '''
        '''
        pass

if __name__ == '__main__':
    # Using WPILib we can run our robot with the following
    # command
    wpilib.run(Robot)
```