

Converting a Simple Autonomous program to a Command based autonomous program

Converting a Simple Autonomous program to a Command based autonomous program

This document describes how to rewrite a simple autonomous into a command based autonomous. Hopefully, going through this process will help those more familiar with the older simple autonomous method understand the command based method better. By re-writing it as a command based program, there are several benefits in terms of testing and reuse. For this example, all of the logic is abstracted out into functions primarily so that the focus of this example can be on the structure.

The initial autonomous code with loops

```
1:  // Aim shooter
2:  SetTargetAngle();
3:  while (!AtRightAngle()) {
4:      CorrectAngle();
5:      delay(); // Delay to prevent maxing CPU
6:  }
7:  HoldAngle();
8:
9:  // Spin up to Speed
10: SetTargetSpeed();
11: while (!FastEnough()) {
12:     SpeedUp();
13:     delay(); // Delay to prevent maxing CPU
14: }
15: HoldSpeed();
16:
17: // Shoot Frisbee
18: Shoot();
```

The code above aims a shooter, then it spins up a wheel and, finally, once the wheel is running at the desired speed, it shoots the frisbee. The code consists of three distinct actions: aim, spin up to speed and shoot the Frisbee. The first two actions follow a command pattern that consists of four parts:

1. Initialization: Seen in lines 2 & 10, prepares for the action to be performed.

Converting a Simple Autonomous program to a Command based autonomous program

2. Condition: Seen in lines 3 & 11, keeps the loop going while it is satisfied.
3. Execution: Seen in lines 4 & 12, repeatedly updates the code to try to make the condition false.
4. End: Seen in lines 7 & 15, performs any cleanup and final task before moving on to the next action.

The last action seen in line 18 only has an explicit initialize, though depending on how you read it, it can implicitly end under a number of conditions. The most obvious one two in this case are when it's done shooting or when autonomous has ended.

Rewriting it as Commands

```
1: public class AutonomousCommand extends CommandGroup {
2:     public AutonomousCommand() {
3:         addSequential(new Aim());
4:         addSequential(new SpinUpShooter());
5:         addSequential(new Shoot());
6:     }
7: }
```

The same code can be rewritten as a CommandGroup that groups the three actions, where each action is written as it's own command. First, the command group will be written, then the commands will be written to accomplish the three actions. This code is pretty straightforward. It does the three actions sequentially, that is one after the other. Line 3 aims the robot, then line 4 spins the shooter

2up and, finally, line 5 actually shoots the frisbee. The addSequential() method sets it so that these commands run one after the other.

Converting a Simple Autonomous program to a Command based autonomous program

The Aim command

```
1: public class Aim extends Command {
2:     public Aim() {
3:         requires(Robot.turret);
4:     }
5:
6:     protected void initialize() {
7:         SetTargetAngle();
8:     }
9:
10:    protected void execute() {
11:        CorrectAngle();
12:    }
13:
14:    protected boolean isFinished() {
15:        return AtRightAngle();
16:    }
17:
18:    protected void end() {
19:        HoldAngle();
20:    }
21:
22:    protected void interrupted() {
23:        end();
24:    }
25: }
```

As you can see, the command reflects the four parts of the action we discussed earlier. It also has the interrupted() method which will be discussed below. The other significant difference is that the condition in the isFinished() is the opposite of what you would put as the condition of the while loop, it returns true when you want to stop running the execute method as opposed to false. Initializing, executing and ending are exactly the same, they just go within their respective method to indicate what they do.

Converting a Simple Autonomous program to a Command based autonomous program

SpinUpShooter command

```
1: public class SpinUpShooter extends Command {
2:     public SpinUpShooter() {
3:         requires(Robot.shooter);
4:     }
5:
6:     protected void initialize() {
7:         SetTargetSpeed();
8:     }
9:
10:    protected void execute() {
11:        SpeedUp();
12:    }
13:
14:    protected boolean isFinished() {
15:        return FastEnough();
16:    }
17:
18:    protected void end() {
19:        HoldSpeed();
20:    }
21:
22:    protected void interrupted() {
23:        end();
24:    }
25: }
```

The spin up shooter command is very similar to the Aim command, it's the same basic idea.

Converting a Simple Autonomous program to a Command based autonomous program

Shoot command

```
1: public class Shoot extends Command {
2:     public Shoot() {
3:         requires(shooter);
4:     }
5:
6:     protected void initialize() {
7:         Shoot();
8:     }
9:
10:    protected void execute() {} // Do Nothing
11:
12:    protected boolean isFinished() {
13:        return true;
14:    }
15:
16:    protected void end() {} // Do Nothing
17:    protected void interrupted() {
18:        end();
19:    }
20: }
```

The shoot command is the same basic transformation yet again, however it is set to end immediately. In CommandBased programming, it is better to have it's isFinished method return true when the act of shooting is finished, but this is a more direct translation of the original code.

Benefits of the command based approach

Why bother re-writing the code as CommandBased? Writing the code in the CommandBased style offers a number of benefits:

- **Re-Usability** You can reuse the same command in teleop and multiple autonomous modes. They all reference the same code, so if you need to tweak it to tune it or fix it, you can do it in one place without having to make the same edits in multiple places.
- **Testability** You can test each part using tools such as the SmartDashboard to test parts of the autonomous. Once you put them together, you'll have more confidence that each piece works as desired.
- **Parallelization** If you wanted this code to aim and spin up the shooter at the same time, it's trivial with CommandBased programming. Just use AddParallel() instead of AddSequential() when adding the Aim command and now aiming and spinning up will happen simultaneously.

Converting a Simple Autonomous program to a Command based autonomous program

- **Interruptibility** Commands are interruptible, this provides the ability to exit a command early, a task that is much harder in the equivalent while loop based code.