Final Exam Challenge: Autonomous Robotics Group Project

Objective:

In randomized groups, students will program their ROMI robots to autonomously complete a series of tasks: moving in a straight line, turning, and detecting a bump using the accelerometer.

Task Description:

- Move 30 Inches in a Straight Line
- Turn 90 Degrees to the Right
- Move Until the Robot Detects a Bump. Robot should stop after the bump.

Requirements:

- PID Controller: Use PID controllers for precise movement and turning.
- Commands2 Library: Structure the task using the commands2 library.
- Accelerometer Integration: Use the accelerometer to detect the bump.
- Group Collaboration: Groups will be randomized, and each group must collaborate to complete the task.

Example Code Snippet: The following code snippets provide an example of how to structure the autonomous commands using the commands2 library and PID controllers. They won't work, and require some modifications. But they may be helpful in understanding the basic structure.

Move 30 Inches Command:

```python
import commands2
from wpilib.controller import PIDController

class MoveStraight(commands2.CommandBase):
    # Your method names may vary
    def __init__(self, distance, drivetrain):
        super().__init__()
        ... # Some code goes here
        self.pid = PIDController(0.1, 0, 0.1)

    def initialize(self):
        self.drivetrain.reset_encoders()
        self.pid.setSetpoint(self.distance)

    def execute(self):
        current_distance = self.drivetrain.get_distance()
        error = self.distance - current_distance
        output = self.pid.calculate(error)
        # Need to add some code to ake sure that drives straight
        self.drivetrain.arcadeDrive(output, turn)

    def isFinished(self):
        return abs(self.distance - self.drivetrain.get_distance()) < 0.5

    def end(self, interrupted):
        self.drivetrain.arcadeDrive(0, 0)
```

Turn 90 Degrees Command:

```python
import commands2
```

```
from wpilib.controller import PIDController

class Turn(commands2.CommandBase):
    # Your method names may vary
    def __init__(self, angle, drivetrain):
        super().__init__()
        self.angle = angle
        self.drivetrain = drivetrain
        self.pid = PIDController(0.1, 0, 0.1)

    def initialize(self):
        self.initial_angle = self.drivetrain.get_gyro_angle()
        self.pid.setSetpoint(self.angle)

    def execute(self):
        current_angle = self.drivetrain.get_gyro_angle() - self.initial_angle
        error = self.angle - current_angle
        output = self.pid.calculate(error)
        self.drivetrain.set_motor_speeds(output, -output)

    def isFinished(self):
        return abs(self.angle - (self.drivetrain.get_gyro_angle() - self.initial_angle)) < 2.0

    def end(self, interrupted):
        self.drivetrain.set_motor_speeds(0, 0)
```

Move Until Bump Command:

```
import commands2

class MoveUntilBump(commands2.CommandBase):
    # Your method names may vary
    def __init__(self, drivetrain):
        super().__init__()
        self.drivetrain = drivetrain
        self.initial_z = 0

    def initialize(self):
        self.drivetrain.reset_encoders()
        self.initial_z = self.drivetrain.get_accelZ()

    def execute(self):
        self.drivetrain.arcadeDrive(.5, 0)

    def isFinished(self):
        # Check if the Z acceleration exceeds a threshold indicating a bump
        current_z = self.drivetrain.getZ()
        return abs(current_z - self.initial_z) > 1.0  # Adjust threshold as necessary

    def end(self, interrupted):
        self.drivetrain.set_motor_speeds(0, 0)
```

Main Autonomous Command Sequence: This is another way to create a complex command. This is a SequentialCommandGroup that runs the commands in sequence. Then in `robot.py` you can import and then schedule this command in the `autonomousInit` method.

```python
import commands2

class AutonomousSequence(commands2.SequentialCommandGroup):
    def __init__(self, drivetrain, accelerometer):
        super().__init__(
            MoveStraight(30, drivetrain),  # Move 30 inches
            Turn(90, drivetrain),           # Turn 90 degrees to the right
            MoveUntilBump(drivetrain)  # Move until bump is detected
        )
```

Project Presentation:

- Functionality: Demonstrate the robot completing the task.
- Code Explanation: Explain the implementation of PID controllers, use of the commands2 library, and accelerometer integration.
- Collaboration: Discuss the group dynamics and how each member contributed to the project.

Evaluation Criteria:

- Task Completion: Did the robot successfully complete the task?
- Precision: How accurate were the movements and bump detection?
- Code Quality: Is the code well-documented and structured?
- Understanding: Can the group explain their approach and the role of each component?

Groups

- Group 1
  - Eben
  - Sky
- Group 2
  - Cam
  - Zoie
- Group 3
  - Huber
  - Liya
- Group 4
  - Reese
  - Max
  - Nia
- Group 5
  - Raghav
  - Hannah