Group Homework 1: Collaboration with Git, GitHub, and Python

Objective

This assignment is designed to simulate a real-world software development project in the field of robotics. Working in groups of 3-4, you will collaborate to develop a Python-based software for controlling a robot with a differential drivetrain, implementing odometry tracking, and managing a shooter mechanism. The project emphasizes teamwork, version control with Git and GitHub, and practical coding skills in a robotics context.

General Instructions

- Teamwork: Form groups of 3-4 students. Assign roles but ensure cross-review across parts for a broader learning experience.
- Version Control: Use Git and GitHub for collaboration. Each team member will work on their component in separate branches, utilizing pull requests for code integration.
- Code Review: Critically review each other's pull requests. Provide constructive feedback and request changes if necessary. Approve pull requests that meet quality standards.
- Integration: Merge feature branches into the main branch upon completion and review.

Part 1: Drivetrain

Develop the code to control the robot's differential drive. This system allows the robot to move by varying the speeds of its left and right wheels.

Properties and Methods Implement properties as specified, utilizing the Motor_Controller class for motor objects. The **init** method should initialize motors and set wheel base, wheel radius, and gearing ratio. Implement methods to set speeds, stop the robot, and calculate linear speeds and individual wheel speeds.

Recommendations

- Code Structure: Ensure your code is modular, making it easy to read, maintain, and test.
- Testing: Include unit tests for your methods to verify correct behavior under various conditions.

Part 2: Odometry

Implement odometry to track the robot's position and orientation using the differential drive data.

Properties and Methods Your **init** method must initialize with a drivetrain object. The update method should calculate the robot's current position (x, y) and orientation (theta) based on wheel movements. Research differential drive odometry to accurately implement the update function.

Recommendations

- Mathematical Modeling: Be meticulous with your mathematical models and calculations for accurate position tracking.
- Simulation: Consider simulating the odometry system to test its accuracy before integrating it with the physical robot.

Part 3: Shooter

Create a subsystem to control a shooter mechanism, including flywheel speed and shooting angle.

Properties and Methods Use Motor_Controller objects for controlling the shooter's components. Implement methods to set the flywheel speed and shooter angle.

Recommendations Experimentation: Experiment with different speeds and angles to determine optimal settings for various shooting tasks.

Additional Notes

- Documentation: Document your code thoroughly to help your teammates and future selves understand your logic.
- Git Practice: Make use of Git features like branching, merging, and pull requests to gain familiarity with collaborative coding practices.
- Learning Opportunity: This assignment is not just about coding but also about learning to work as part of a software development team. Take the opportunity to learn from each other.

Deliverables

Detailed Class Specifications

Part 1: Drivetrain Class

Properties

- left_motor_controller: An instance of the Motor_Controller class to control the left motor.
- right_motor_controller: An instance of the Motor_Controller class to control the right motor.
- wheel_base: The distance between the two wheels on the robot (in units of your choice, but consistent across your project).
- wheel_radius: The radius of the wheels (in units of your choice, but consistent across your project).
- gearing_ratio: The ratio of motor turns to wheel turns. This determines how many rotations the motor has to make to complete one full rotation of the wheel.

Methods

- **init**(self, wheel_base, wheel_radius, gearing_ratio): Constructor to initialize the drivetrain with the wheel base, wheel radius, and gearing ratio. It should also instantiate the left and right motor controllers.
- set_speeds(self, left_speed, right_speed): Sets the speed of the left and right motors. The speeds are numbers between -1 and 1, where -1 is full speed backward, 1 is full speed forward, and 0 is stop.
- stop(self): Stops both the left and right motors.
- get_linear_speed(self): Calculates and returns the linear speed of the robot based on the current speeds of the left and right motors, the wheel radius, and the gearing ratio.
- get_left_speed(self): Returns the current speed setting of the left motor.
- get_right_speed(self): Returns the current speed setting of the right motor.

Part 2: Odometry Class

Properties

- drivetrain: A reference to an instance of the Drivetrain class.
- x: The current x-coordinate of the robot on the field.
- y: The current y-coordinate of the robot on the field.
- theta: The current orientation (angle) of the robot in radians.

Methods

- **init**(self, x, y, drivetrain): Constructor to initialize the odometry with a reference to the drivetrain object.
- update(self): Updates the robot's position (x, y) and orientation (theta) based on the current speeds of the left and right motors. This method requires applying the differential drive odometry formulas.
- get_position(self): Returns the current position and orientation of the robot as a tuple (x, y, theta).

Part 3: Shooter Class

Properties

- motor_controller_flywheel: An instance of the Motor_Controller class to control the flywheel motor.
- motor_controller_angle: An instance of the Motor_Controller class to adjust the angle of the shooter.

- motor_controller_feeder: An instance of the Motor_Controller class to control the feeder mechanism that introduces balls to the shooter.
- flywheel_speed: The current speed of the flywheel.
- angle: The current angle of the shooter mechanism.
- feeder_speed: The speed of the feeder mechanism.
- loaded: A boolean that indicates if the shooter is loaded with a ball.

Methods

- **init**(self): Constructor to initialize the shooter mechanism. This should create motor controllers for the flywheel, angle adjustment, and feeder.
- set_flywheel_speed(self, speed): Sets the speed of the flywheel. The speed parameter should dictate how fast the flywheel spins.
- set_angle(self, angle): Sets the angle of the shooter mechanism. The angle parameter should specify the shooting angle.
- set_feeder_speed(self, speed): Sets the speed of the feeder mechanism. This controls how fast balls are introduced to the shooter.
- set_loaded(self): Sets the loaded flag to True when a ball is loaded in the shooter.
- set_unloaded(self): Sets the loaded flag to False when the shooter is empty.