Drone Movement Controller Code Explanation

Overview

This code defines a Unity C# class `droneMovementController` that manages and controls the movement, stabilization, and overall behavior of a simulated drone. The class implements physics-based stabilization, target tracking, and rotor control mechanisms, as well as debugging tools for visualizing and logging data.

1 Physical Parts and Related Functions

This section sets up the drone's physical components, including sensors, rotors, and stabilization mechanisms:

1.1 Sensors

The following sensors are modeled as public objects and must be linked to the corresponding Unity GameObjects or scripts:

- `Gyro`: Measures rotational velocities.
- `Accelerometer`: Measures linear accelerations.
- `Barometer`: Measures altitude.
- `GPS`: Measures global position.
- `Magnetometer`: Measures orientation relative to magnetic north.

1.2 Rotors

Four rotors control the drone's movement:

- `helixV1, helixV2, helixO1, helixO2`: Represent the four propellers of the drone.

Their rotation power is adjusted to control the drone's movement.

1.3 PID Controllers

PID controllers are used for stabilization:

- `yawPID, rollPID, pitchPID, xPID, yPID, zPID`: Each PID controller handles a specific stabilization task.

1.4 Helper Functions

- `applyTorque(amount)`: Simulates torque affecting the drone.
- `torqueGeneratedBy(rotor r)`: Computes the torque generated by a single rotor.
- `denormalizePower(pow)`: Converts normalized power values (range $[0, 1]$) to real-world rotor power.
- `denormalizeTorque(pow)`: Converts normalized power values to torque.
- `keepOnRange01(num)`: Keeps a value within the range $[0, 1]$.

2 Target Settings

Defines the drone's target states or positions:

- `idealPitch, idealRoll, idealYaw`: Represent the desired orientation of the drone.
- `targetX, targetY, targetZ`: Represent the desired spatial positions.
- `routePosition`: A private variable defining a route-related position.
- `lookingAtPoint`: A private variable defining the point the drone should face.
- `stayOnFixedPoint`: A boolean flag indicating whether the drone should stabilize itself or follow a target.

3 Internal Inputs

This section allows external algorithms to modify the drone's behavior:

- `setConsts(vVel, vAcc, aVel, aAcc, yVel, orVel, orAcc)`: Allows external algorithms to modify velocity and acceleration constants.
- `setKs(yPID, zPID, xPID, pitchPID, rollPID, yawPID)`: Allows external algorithms to assign new PID controllers.

4 Rotor Outputs

Controls the power of the rotors:

- `pV1, pV2, pO1, pO2`: Store normalized power values for each rotor (range $[0, 1]$).
- Functions like `modifyAllRotorsRotation`, `modifyRollRotorsRotation`, etc., adjust rotor power to achieve desired movements.

5 Stabilization Algorithms

The following algorithms stabilize the drone in six directions:

5.1 Vertical Stabilization (`yStabilization`)

- Uses the barometer to stabilize the drone's height.
- Calculates the error between the target altitude (y_{target}) and the current altitude (y_{current}).
- Adjusts rotor power using PID output.

5.2 Roll Stabilization (`rollStabilization`)

- Uses gyroscopic data to stabilize the drone's roll.
- Calculates the error between the target roll (ϕ_{target}) and current roll (ϕ_{current}).
- Adjusts rotor power using PID output.

5.3 Pitch Stabilization (`pitchStabilization`)

- Stabilizes the drone's pitch using gyroscopic data.
- Calculates the error between the target pitch (θ_{target}) and current pitch (θ_{current}).

5.4 Yaw Stabilization (`yawStabilization`)

- Stabilizes the drone's orientation (yaw) using magnetometer data.
- Adjusts rotor power to minimize yaw errors.

5.5 Z Stabilization (`zStabilization`)

- Stabilizes forward-backward movement using accelerometer data.
- Returns the pitch error for further corrections.

5.6 X Stabilization (`xStabilization`)

- Stabilizes side-to-side movement using accelerometer data.
- Returns the roll error for further corrections.

6 Debugging Tools

- `lineDrawer`: Visualizes direction vectors in Unity's scene view for debugging.
- `dataSaver`: Logs performance data for analysis and testing.

7 Lifecycle Functions

Standard Unity lifecycle methods:

- `Start()`: Initializes PID controllers and debugging tools.
- `Update()`: Saves debugging data if `save` is true.
- `FixedUpdate()`: Runs every fixed interval to:
  - Calculate the drone's target positions and orientations.
  - Call stabilization algorithms to adjust rotor power.
  - Apply torque and normalize rotor power.

8 Workflow of the Drone Controller

1. **Initialization:** PID controllers and debugging tools are set up.
2. **Target Updates:** The drone recalculates its targets in `FixedUpdate`.
3. **Stabilization:** Algorithms adjust rotor power based on sensor readings.
4. **Rotor Control:** Normalized power is applied to the rotors.
5. **Debugging:** Logs and visualizations provide insights for debugging.