minilab1_hardware.py

```python
"""
IMPORTANT NOTE:
    The instructions for completing this template are inline with the code. You can
    find them by searching for: "TODO:"
"""

import signal
import time
from collections import deque

import dxl
import matplotlib.pyplot as plt
import numpy as np
from dxl import DynamixelMode, DynamixelModel
from numpy.typing import NDArray

from mechae263C_helpers.minilabs import FixedFrequencyLoopManager, ExponentialFilter


class PIDPositionController:
    """
    This class manages a PID Position Controller
    """

    def __init__(
        self,
        motor: dxl.DynamixelMotor,
        proportional_gain: float,
        integral_gain: float = 0.0,
        derivative_gain: float = 0.0,
        ang_set_point_deg: float = 90.0,
        control_freq_Hz: float = 100.0,
        max_duration_s: float = 2.0,
    ):
        self.max_duration_s = float(max_duration_s)
        self.motor = motor
        self.should_continue = True

        # `FixedFrequencyLoopManager` trys keep loop at a fixed frequency
        self.loop_manager = FixedFrequencyLoopManager(control_freq_Hz)

        # Set the position set-point in units of degrees
        self.position_set_point_deg = max(
            min(ang_set_point_deg, motor.model_info.max_angle_deg), 0.0
        )

        # Set PID gains
        self.proportional_gain = float(proportional_gain)
```

```
        self.integral_gain = float(integral_gain)
        self.derivative_gain = float(derivative_gain)

        signal.signal(signal.SIGINT, self.signal_handler)
        signal.signal(signal.SIGTERM, self.signal_handler)

        self.error = 0.0
        self.error_integral = 0.0
        self.error_derivative = 0.0
        self.error_window = deque(maxlen=3)

        self.filter = ExponentialFilter(0.97, num_warmup_time_steps=0)

        self._position_history = deque()
        self._timestamps = deque()

        # Put motor in "home" posiiton
        self.motor.set_mode(DynamixelMode.Position)
        self.motor.enable_torque()
        self.motor.angle_deg = 0.0
        while abs(self.motor.angle_deg) > 0.8:
            self.motor.angle_deg = 0.0
            time.sleep(0.5)

        self.motor.disable_torque()
        time.sleep(0.5)

        # PWM Mode (i.e. voltage control)
        self.pwm_limit = float(self.motor.motor_info.pwm_limit)
        motor.set_mode(DynamixelMode.PWM)

    @property
    def position_history(self) -> NDArray[np.double]:
        return np.asarray(self._position_history)

    @property
    def timestamps(self) -> NDArray[np.double]:
        t = np.asarray(self._timestamps)
        t -= t[0]
        return t

    def start(self):
        self.motor.enable_torque()
        curr_time = time.time_ns()
        while self.should_continue:
            # ------------------------------------------------------------------------
            # Step 1 - Get Feedback
            # ------------------------------------------------------------------------
            ang_deg = self.motor.angle_deg

```

```python
            self._position_history.append(ang_deg)  # Save for plotting
            self._timestamps.append(time.time())  # Save for plotting

            # ------------------------------------------------------------------------
            # Step 2 - Update PID Terms
            # ------------------------------------------------------------------------
            self.error = self.position_set_point_deg - ang_deg

            self.error_window.append(self.error)

            prev_time = curr_time
            curr_time = time.time_ns()

            self.error_integral += self.error * (curr_time - prev_time) / 1e9

            if len(self.error_window) == self.error_window.maxlen:
                self.error_derivative = (
                    (
                        3 * self._position_history[-1]
                        - 4 * self._position_history[-2]
                        + self._position_history[-3]
                    )
                    / 2
                    / ((curr_time - prev_time) / 1e9)
                )
                self.error_derivative = -self.filter(self.error_derivative)
            elif len(self.error_window) == 2:
                self.error_derivative = -self.filter(
                    (self._position_history[-1] - self._position_history[-2])
                    / ((curr_time - prev_time) / 1e9)
                )
            else:
                self.error_derivative = self.filter(0.0)

            # ------------------------------------------------------------------------
            # Step 3 - Check termination criterion
            # ------------------------------------------------------------------------
            # Stop after 2 seconds
            if self._timestamps[-1] - self._timestamps[0] > self.max_duration_s:
                self.motor.disable_torque()
                return

            # ------------------------------------------------------------------------
            # Step 4 - Calculate and send command
            # ------------------------------------------------------------------------
            pwm_command = (
                self.proportional_gain * self.error
                + self.integral_gain * self.error_integral
                + self.derivative_gain * self.error_derivative
            )
```

```python

            # Saturate control action (pwm duty cycle in the range [-100.0, 100.0])
            pwm_command = max(min(pwm_command, 100.0), -100.0)
            self.motor.pwm_percentage = pwm_command

            # Print current position in degrees
            print("Current Position:", self.motor.angle_deg)

            self.loop_manager.sleep()

        self.motor.disable_torque()

    def stop(self):
        self.should_continue = False
        time.sleep(self.loop_manager.period_s)

    def signal_handler(self, *_):
        self.stop()


if __name__ == "__main__":
    # Create `DynamixelIO` object to store the serial connection to U2D2
    #
    # TODO: Replace "..." below with the correct serial port found from Dynamixel Wizard
    #
    # Note: You may need to change the Baud Rate to match the value found from
    #       Dynamixel Wizard
    dxl_io = dxl.DynamixelIO(
        device_name="COM6",       # Port
        baud_rate=57_600,
    )

    # Create `DynamixelMotorFactory` object to create dynamixel motor object
    motor_factory = dxl.DynamixelMotorFactory(
        dxl_io=dxl_io,
        dynamixel_model=DynamixelModel.MX28
    )

    # TODO: Replace "..." below with the correct Dynamixel ID found from Dynamixel Wizard
    motor = motor_factory.create(4)

    # Make controller
    # TODO: Replace all "..." below with your selected choices of gains.
    controller = PIDPositionController(
        motor=motor,
        proportional_gain=10,
        integral_gain=2,
        derivative_gain=1,
        control_freq_Hz=100,
    )
```

```python

    # Start control loop
    controller.start()

    # -----------------------------------------------------------------------------------
    # Plot Results
    # -----------------------------------------------------------------------------------
    gains = (
        controller.proportional_gain,
        controller.integral_gain,
        controller.derivative_gain,
    )
    fig_file_name = f"p{gains[0]}-i{gains[1]}-d{gains[2]}.pdf"

    # Create figure and axes
    fig = plt.figure(figsize=(10, 5))
    ax = fig.add_subplot(111)

    # Plot setpoint of 90.0 angle trajectory (with label)
    ax.set_title(
        f"Motor Angle vs Time ($K_p$={gains[0]}, $K_i$={gains[1]}, $K_d$={gains[2]})"
    )
    ax.set_xlabel("Time [s]")
    ax.set_ylabel("Motor Angle [deg]")

    # Plot setpoint of 90.0 angle trajectory (with label)
    ax.axhline(
        controller.position_set_point_deg, ls="--", color="red", label="Setpoint"
    )
    # Plot motor angle trajectory (with label)
    ax.plot(
        controller.timestamps,
        controller.position_history,
        color="black",
        label="Motor Angle Trajectory",
    )
    ax.legend()

    fig.savefig(fig_file_name)

    plt.show()

```