```python
import yaml
from time import sleep

from pyfirmata import Arduino, util, STRING_DATA


def load_config(file):
    """
    :param file:
    :return:
    """
    try:

        with open(file) as file:
            return yaml.load(file, Loader=yaml.FullLoader)
    except FileNotFoundError as fe:
        exit(f'Could not find {file}')

    except Exception as e:
        exit(f'Encountered exception...\n {e}')


def read_angle(initial, **kwargs) -> float:
    """
    :param initial:
    :param kwargs:
    :return rotation_angle:
    """
    rotation_angle = _map(initial, kwargs['in_min'], kwargs['in_max'], 0.00, 90.00)
    board.send_sysex(STRING_DATA, util.str_to_two_byte_iter('   '))
    board.send_sysex(STRING_DATA, util.str_to_two_byte_iter(f'{rotation_angle} degrees'))
    return rotation_angle


def set_pwm(**kwargs) -> float:
    """
    :param kwargs:
    :return pwm_signal:
    """
    pwm_signal = _map(
        kwargs['run_angle'],
        kwargs['in_angle_min'],
        kwargs['in_angle_max'],
        kwargs['pwm_min'],
        kwargs['pwm_max']
    )
    return pwm_signal


def _map(x, in_min, in_max, out_min, out_max) -> float:
    """
    :param x:
    :param in_min:
    :param in_max:
    :param out_min:
    :param out_max:
    :return:
    """
    print(x)
    return round((x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min, 4)


if __name__ == '__main__':

    # run constants
    RUN_ENABLED: bool = False
    NEW_PWM_SIGNAL: bool = False
    PWM_SIGNAL: float = .00
    RUN_ERROR: int = 0
    RUN_MODE_SET: bool = False
    pot_val = 0.00

    # parse configs from config.yml file
    parsed_config = load_config('config.yml')

    # load configs
    RUN_ANGLE = parsed_config['copter_settings'].get('RUN_ANGLE')
    MIN_ANGLE = parsed_config['copter_settings']['run_settings'].get('IN_ANGLE_MIN')
    MAX_ANGLE = parsed_config['copter_settings']['run_settings'].get('IN_ANGLE_MAX')
    RUN_MODE = parsed_config['copter_settings']['run_mode'].get('CW')
    ANALOG_READ_MIN = parsed_config['copter_settings']['run_settings'].get('IN_ANALOG_READ_VAL_MIN')
    ANALOG_READ_MAX = parsed_config['copter_settings']['run_settings'].get('IN_ANALOG_READ_VAL_MAX')
    PWM_MIN = parsed_config['copter_settings']['run_settings'].get('PWM_OUT_MIN')
    PWM_MAX = parsed_config['copter_settings']['run_settings'].get('PWM_OUT_MAX')
    PWM_PIN = parsed_config['copter_settings']['run_settings'].get('
PWM_OUT_PIN')
    A_READ_PIN = parsed_config['copter_settings']['run_settings'].get('A_READ_PIN')
    PORT = parsed_config['copter_settings']['run_settings'].get('PORT')
    RUN_PIN = parsed_config['copter_settings']['run_settings'].get('RUN_MODE_PIN')

    # initializing arduino board with port
    board = Arduino(PORT.lower())

    #  args to set pwm output
    SET_PWM_ARGS = dict(
        run_angle=RUN_ANGLE,
        in_angle_min=MIN_ANGLE,
        in_angle_max=MAX_ANGLE,
        pwm_min=PWM_MIN,
        pwm_max=PWM_MAX
    )

    print('-' * 20, 'preparing arduino uno board to load in 100 milliseconds', '-' * 20)

    # sending starting message over the serial
    board.send_sysex(STRING_DATA, util.str_to_two_byte_iter(' Initializing '))

    # waiting for the board to be ready
    sleep(.1)
    board.send_sysex(STRING_DATA, util.str_to_two_byte_iter(' Board is Ready '))

    it = util.Iterator(board)
    it.start()
    board.analog[A_READ_PIN].enable_reporting()

    # setting  digital pin 3 as a pwm output
    PWM = board.get_pin(f'd:{PWM_PIN}:p')

    # # setting digital pin 8 as a switch to control the direction of motor rotation
    # CW = board.get_pin('d:8:o')

    while True:

        if not RUN_ENABLED:
            print('-' * 20, 'waiting for analog response', '-' * 20)
            sleep(2.5)

        # reading analog value from the potentiometer to detect the angle
        ANALOG_READ_VAL = board.analog[A_READ_PIN].read()
        print(ANALOG_READ_VAL, ' value of analog read')
        # # board.send_sysex(STRING_DATA, util.str_to_two_byte_iter(f'{ANALOG_READ_VAL}'))
        # if a_value := ANALOG_READ_VAL:
        #     # if a_value < ANALOG_READ_MIN:
        #     #     ANALOG_READ_VAL = a_value
        #     RUN_ENABLED = True
        # else:
        #     sleep(1.5)
        #     ANALOG_READ_VAL = board.analog[A_READ_PIN].read()
        #
        # # if the motor is to rotate in counterclockwise
        # print('clockwise ', RUN_MODE)
        if not RUN_MODE_SET:
            board.digital[RUN_PIN].write(int(RUN_MODE))
            RUN_MODE_SET = True

        # starting the motor when the system configurations are done
        if not NEW_PWM_SIGNAL:
            board.send_sysex(STRING_DATA, util.str_to_two_byte_iter(' motor starting '))
            #
            PWM_SIGNAL = set_pwm(**SET_PWM_ARGS)
            print(PWM_SIGNAL, 'pwm signal')
            PWM.write(PWM_SIGNAL)
            RUN_ERROR += 1
            sleep(5.5)
            RUN_ENABLED = True

        # read the angle achieved by the motor through the potentiometer
        print(ANALOG_READ_VAL, ' analog value when reading the angle')
        angle = read_angle(
            ANALOG_READ_VAL,
            **dict(
                in_min=ANALOG_READ_MIN,
                in_max=ANALOG_READ_MAX
            )
        )
        print('*' * 50)
        print(angle, ' degrees')
        print('*' * 50)
        # """
        # Handling feedback when the motor has not achieved the required angle,
        # error can be positive or negative depending on the value of pwm
```

```
signal sent,
        # to the driver from the microcontroller,
        # this handles the error correction to a much steady value.
        # """
        if RUN_ERROR > 0:
            if RUN_ANGLE != angle:
                if diff := (RUN_ANGLE - angle):
                    print('-' * 20, f'correcting error of {diff} degrees', '-' * 20)
                    if diff < 0:
                        SET_PWM_ARGS.update(run_angle=diff)
                        new = set_pwm(**SET_PWM_ARGS)
                        print(new, 'error when diff lt 0')
                        PWM_SIGNAL -= new
                    elif diff > 0:
                        SET_PWM_ARGS.update(run_angle=diff)
                        new = set_pwm(**SET_PWM_ARGS)
                        print(new, 'error when diff gt 0')
                        PWM_SIGNAL += new
                    SET_PWM_ARGS.update(run_angle=RUN_ANGLE)
                    # NEW_PWM_SIGNAL = True
                    print(SET_PWM_ARGS)
                    print(PWM_SIGNAL, " corrected")
                    # send a new signal with a feedback included
                    # if not abs(PWM_SIGNAL) > 1:
                    # PWM.write(abs(PWM_SIGNAL))

                    #     # sleep to allow the motor to achieve the angle
                    #     sleep(3.0)

```