PID 제어를 통한 드론 자율 주행 코딩 활동

•연구학생: 20401 강하윤,
20417 이주현

연구내용

1. 활동 내용 및 방법 [작성자 : 강하윤]
- 단계별 활동 과정 정리 (실습 1, 2, 3 …처럼 번호를 붙여 구체적으로 작성)
- 사용한 코드, 회로도, 장치 설명 (사진 첨부 가능)
- 주요 개념 정리 (예: PID 제어, 센서 원리, 통신 규약 등)
-

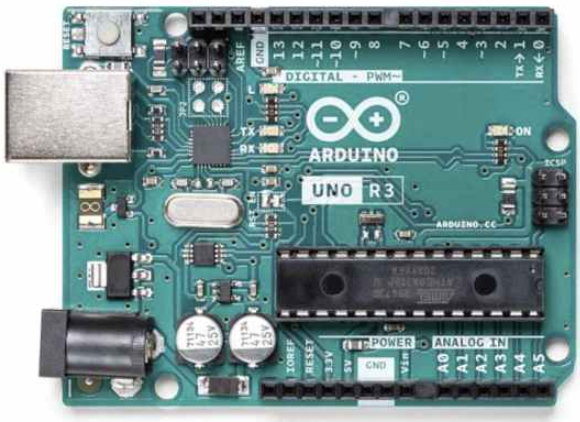
2. 실험 및 탐구 결과 [작성자 : 이주현]
- 활동에서 얻은 데이터, 관찰 결과, 시뮬레이션 결과
- 그림, 표, 그래프 활용 권장
- 코드 실행 화면, 시리얼 모니터 출력 예시 첨부 가능

3. 분석 및 토의 [작성자 : 이주현]
- 결과가 의미하는 바 해석
- 예상과 다른 결과가 나온 경우 원인 분석
- 문제 해결을 위해 시도한 방법

4. 결론 및 성찰 [작성자 : 강하윤]
- 이번 활동을 통해 배운 점
- 어려웠던 점과 해결 과정
- 앞으로 확장할 수 있는 탐구 방향

1. 활동 내용 및 방법

1-1. 사용한 장치 설명



- 아두이노

- 드론

- 아두이노(Arduino) 및 제어기(Flight Controller)

아두이노 기반 드론 키트와 상용 FC 보드(Ardupilot, Multiwii 등)를 활용하여 비행 제어를 수행한다.

각종 센서(자이로, 가속도, 지자계, GPS, 기압, 초음파 등)에서 신호를 받아 드론의 자세와 이동을 제어한다.

- 모터(Coreless Motor 37)와 변속기(ESC)

FC에서 제어 신호(PWM)를 받아 변속기가 모터의 속도를 제어한다.

드론에 사용되는 모터는 Brushed Motor, BLDC Motor, Coreless Motor 등이 있으며, 실습에는 Coreless Motor가 사용된다.

Coreless Motor는 철심이 없는 구조로 가볍고 반응 속도가 빠르며, 효율은 약 70~80% 정도이다.

- 센서(Sensor)

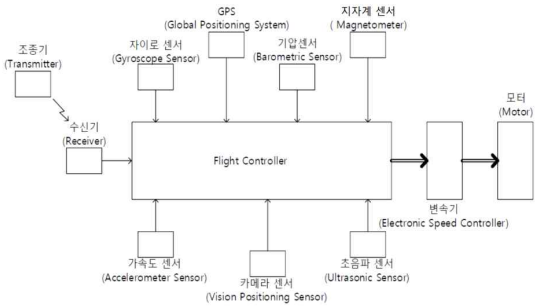
자이로 센서: 드론의 각속도(회전 속도) 측정
가속도 센서: 드론의 가속도 및 기울기 측정
지자계 센서: 지구 자기장을 이용하여 방향 파악
GPS: 위성 신호로 드론 위치 추적
기압 센서: 기압 변화를 통해 고도 계산
초음파 센서: 지면과의 거리를 측정

- 조종기 및 통신 장치

조종기(Transmitter)에서 신호를
송신 → 수신기(Receiver)에서 수신 → FC로 전달

드론의 상태 정보는 다시 송신기를 통해 조종기로 전달된다.

1-2. 회로도(계통도)



실제 드론은 조종기 → 수신기 → 제어기(FC) → 변속기(ESC) → 모터 순으로 신호가 흐른다.

센서(IMU, GPS, 초음파 등)는 I2C 통신 방식을 통해 제어기에 연결된다.

Flight Controller는 모터 제어와 동시에 센서 데이터를 이용하여 안정적인 비행을 유지한다.

1-3. 주요 개념 정리

- 드론에 작용하는 힘

양력(Lift): 공기의 흐름을 이용해 상승하는 힘
중력(Weight): 지구 중심으로 작용하는 힘
추력(Thrust): 프로펠러 회전으로 발생하는 추진력
항력(Drag): 이동 반대 방향으로 작용하는 저항력

- Roll, Pitch, Yaw, Throttle

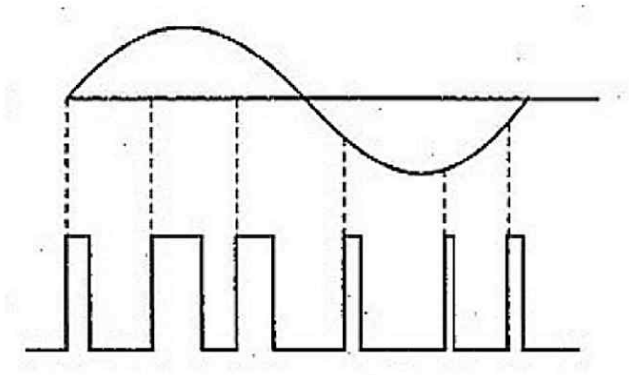
Roll(X축): 좌우 기울기
Pitch(Y축): 앞뒤 기울기
Yaw(Z축): 수평 회전
Throttle: 모터 속도 제어(상승/하강)

- I2C 통신

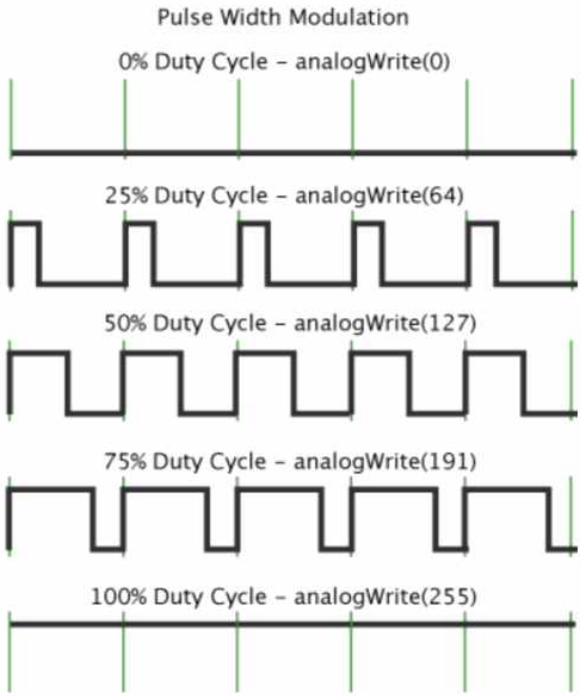
센서와 제어기를 연결하는 직렬 통신 방식

SDA(데이터), SCL(클록) 두 개의 선만으로 다중 센서 연결 가능

- PWM 제어



디지털 신호로 아날로그 신호를 모사하는 방식



Duty 비율에 따라 모터 속도를 제어

- PID 제어

드론의 회전 각도를 0으로 유지하거나 다른 위치로 이동시키기 위해 목표 각도 설정 필요 → 현재 각도와 목표 각도의 오차가 0이 되도록 모터를 조정하는 PID 제어

P(비례): 현재 오차 크기에 비례해 보정
I(적분): 누적 오차를 고려하여 장기적 보정
D(미분): 오차 변화율을 반영하여 진동 억제

PID 제어를 통해 드론이 안정적으로 비행 가능

1-4. 단계별 활동 과정 정리

실습 1

실습 1-1 : PWM 이해하기

ONE_PERIOD와 HIGH_TIME 값을 변경하면서 모터 회전 변화를 관찰

Duty Cycle은 고정하고 주기만 변화시켰을 때 → 주파수가 낮아질수록 모터의 동작이 부자연스러워짐
Duty Cycle을 변화시켰을 때 → Duty Cycle이 커질수록 모터 회전력이 증가

실습 1-2 : analogWrite 함수 사용

VALUE 값을 변경하면서 모터 속도 변화를 관찰

실습 1-3 : 모든 모터 구동하기

드론에 연결된 4개의 모터를 순서대로 구동하고 정지시키는 실습을 진행, 각 모터가 독립적으로 제어되는지 확인

실습 1-4 : 모터 속도 제어 및 통합 구동

PWM Duty Cycle을 변화시켜 모터 속도를 세밀하게 조절, 4개의 모터를 동시에 제어하여 드론의 균형 잡힌 출력 상태를 실험적으로 확인

실습 2

실습 2-1 : MSP 입력 받기

1. 드론과 다두이노(Daduino) 앱을 연결
2. 시리얼 모니터 실행, 통신 속도 115200으로 맞춤
3. 앱에서 조종 신호를 보낸 뒤, 아두이노에서 출력되는 MSP 데이터 확인

실습 2-2 : MSP 메시지에서 roll, pitch, yaw, throttle 값 추출

패킷 중 필요한 제어 값만 추출하기 위해
'$'로 시작하는 패킷을 기준으로 cnt_msg 카운터를 증가시켜 특정 위치 데이터만 출력.

실습 2-3 : 블루투스 모듈로 throttle 값 직접 입력

블루투스 앱에서 throttle 값을 직접 입력해 모터 출력 조정.

실습 3

실습 3-1: 자이로 센서값 읽어보기

1. 아두이노에서 I2C 통신으로 MPU-6050 데이터 레지스터 값 읽기.
2. GYRO_XOUT_H, GYRO_XOUT_L 등 16비트로 합쳐서 GyX, GyY, GyZ 값 획득.
3. 시리얼 모니터로 출력해 범위 확인

확인한 문제점:
1. MPU-6050에서 읽은 값은 °/s 단위가 아니라 원시값, 드론 제어에는 회전 각도가 필요 → 바로 쓰면 단위 불일치 → 실제 드론 제어 불가, 따라서 보정(단위 환산) 필요
2. 누적 오차 발생: 시간이 지남에 따라 점점 편차가 커짐 → 보정 필요

실습 3-2: 자이로 센서값 보정하기

1. 단위 변환 (°/s)
센서 출력 범위: -32768 ~ 32767
실제 범위: ±250 °/s

변환식:

$$\text{각속도}(^\circ/\text{s}) = \frac{\text{자이로 값}}{131}$$

2. 회전 각도 계산
각속도(ω) → 각도(θ) 변환식:

$$\Delta\theta = \omega \times \Delta t$$

아두이노에서 micros() 함수 사용해 Δt 계산
적분을 통해 누적하여 현재 회전 각도 추정

실습 4

실습 4-1: Roll에 대한 비례제어(P)

Kp를 1.0으로 설정한 드론을 손에 잡은 채 모터를 구동시킨 후 기울어짐에 따라 모터 관찰 → 드론을 기울였을 때 기울어진 쪽의 모터 세기 증가

실습 4-2: Roll에 대한 비례, 미분 제어(PD)

Kp, Kd를 각각 1.0으로 설정한 드론을 손에 잡은 채 모터를 구동시킨 후 기울어짐에 따라 모터 관찰 → 드론을 기울였을 때 기울어진 쪽의 모터 세기가 증가했다가 살짝 감소

실습 4-3: Roll에 대한 비례, 미분, 적분 제어(PID)

Kp, Kd, Ki를 각각 1.0으로 설정한 드론을 손에 잡은 채 모터를 구동시킨 후 기울어짐에 따라 모터 관찰 → 드론을 기울였을 때 기울어진 쪽의 모터 세기가 점점 증가, 반대 방향으로 기울였을 때 기존 모터의 세기가 다시 감소

코드 설명

```
#include <Wire.h>

void setup() {
  Serial1.begin(115200);

  Wire.begin();
  Wire.setClock(400000);
```

- I2C를 통해 MPU-6050 센서와 통신. 시리얼 통신으로 외부 입력(키보드, 조종기 등) 받을 준비.

```
  Wire.beginTransmission(0x68);
  Wire.write(0x6b);
  Wire.write(0x0);
  Wire.endTransmission(true);
}
```

- MPU-6050 초기화. 0x6B 레지스터(PWR_MGMT_1)를 0으로 세팅 → 센서 활성화.

```
int throttle = 0;
void loop(){
    Wire.beginTransmission(0x68);
    Wire.write(0x43);
    Wire.endTransmission(false);
    Wire.requestFrom(0x68,6,true);
```

- 0x43 레지스터(GyX)부터 연속 6바이트 읽음 → X, Y, Z축 회전 각속도 값.

```
    int16_t GyXH = Wire.read();
    int16_t GyXL = Wire.read();
    int16_t GyYH = Wire.read();
    int16_t GyYL = Wire.read();
    int16_t GyZH = Wire.read();
    int16_t GyZL = Wire.read();
    int16_t GyX = GyXH << 8 | GyXL;
    int16_t GyY = GyYH << 8 | GyYL;
    int16_t GyZ = GyZH << 8 | GyZL;
```

- 16비트 값으로 합치기.

```
    static int32_t GyXSum = 0, GyYSum = 0, GyZSum = 0;
    static double GyXOff = 0.0, GyYOff = 0.0, GyZOff = 0.0;
    static int cnt_sample = 1000;
```

- 처음 1000번 읽은 값으로 평균을 내서 자이로 드리프트 오프셋 계산. 초기 샘플링 동안은 모터 제어나 PID 계산 없이 오프셋만 계산.

```
    if (cnt_sample>0){
        GyXSum += GyX; GyYSum += GyY; GyZSum += GyZ;
        cnt_sample--;
        if (cnt_sample ==0){
            GyXOff = GyXSum / 1000.0;
            GyYOff = GyYSum / 1000.0;
            GyZOff = GyZSum / 1000.0;
        }
        delay(1);
        return;
    }
    double GyXD = GyX - GyXOff;
    double GyYD = GyY - GyYOff;
    double GyZD = GyZ - GyZOff;
    double GyXR = GyXD / 131; //회전 각속도 --> 드론의 회전 각도를 구할 때뿐만 아니라 미분 제어에서도 사용됨
    double GyYR = GyYD / 131; //회전 각속도 --> 드론의 회전 각도를 구할 때뿐만 아니라 미분 제어에서도 사용됨
    double GyZR = GyZD / 131; //회전 각속도 --> 드론의 회전 각도를 구할 때뿐만 아니라 미분 제어에서도 사용됨
```

- 센서 raw 값 → °/s 단위 변환.
131은 ±250°/s 범위 기준 센서 변환 상수.

```
    static unsigned long t_prev = 0;
    unsigned long t_now = micros();
    double dt = (t_now - t_prev)/1000000.0;
    t_prev = t_now;
```

```
  //현재 드론에 대한 각도를 계산하는 식 코드 작성
    static double AngleX = 0.0, AngleY = 0.0, AngleZ = 0.0;
    AngleX += GyXR * dt;
    AngleY += GyYR * dt;
    AngleZ += GyZR * dt;
```

- 시간 간격 계산 후 각속도를 적분 → 현재 드론 각도 추정.
throttle=0일 때 지표면에 도달 → 각도 초기화.

```
  //throttle이 0일때 지표면에 도달하므로 0인 순간에 각도를 0으로 초기화
    if (throttle ==0) {AngleX = AngleY = AngleZ =0.0;}
```

```
  //제어를 해줄 목표 각도 변수 선언
    static double tAngleX = 0.0, tAngleY = 0.0, tAngleZ = 0.0;
```

```
    //오차값을 정의해준다. 목표 각도 - 현재 드론의 Roll 각도
    double eAngleX = tAngleX - AngleX; //Error AngleX
    double eAngleY = tAngleY - AngleY; //Error AngleY
    double eAngleZ = tAngleZ - AngleZ; //Error AngleZ
```

- 사용자 입력(WASD, QE 등)에 따라 목표 각도 설정.
실제 각도와 목표 각도의 차이를 PID 제어에 사용.

```
    double Kp = 1.0;                //Proportional controller
    //비례 제어에 대한 균형힘 작성 (비례 상수 * 오차 각도값)
    double BalX = Kp * eAngleX;
    double BalY = Kp * eAngleY;
    double BalZ = Kp * eAngleZ;
```

- 목표 각도와 현재 각도의 차이에 비례한 힘 계산.

```
    double Kd = 1.0;          //Differential controller
    //오차 각도를 시간에 대해 미분해주게 되면 회전 각속도를 의미함 비례 제어에서 정의해준 균형 힘에 더해줌
    BalX += Kd * -GyXR;
    BalY += Kd * -GyYR;
    BalZ += Kd * -GyZR;
    if (throttle == 0) {BalX = BalY = BalZ = 0.0;}
```

- 현재 각속도를 기반으로 회전 속도 상쇄 → 진동 억제.

```
    double Ki = 1.0;          //Integral controller
    static double ResX =0.0, ResY=0.0, ResZ =0.0;

    //오차 값에 대한 시간에 대한 적분은 가로 시간축에 대해 오차값을 곱해서 직사각형 형태로 적분을 구함
    ResX += Ki *eAngleX *dt;
    ResY += Ki *eAngleY *dt;
    ResZ += Ki *eAngleZ *dt;
    if(throttle ==0) ResX=ResY =ResZ =0.0;
    BalX += ResX;
    BalY += ResY;
    BalZ += ResZ;
```

- 오차 누적 → 목표 각도 부드럽게 도달.
throttle=0일 때 모두 0으로 초기화 → 지면에서 모터 구동 안 함.

```
    if (Serial1.available() > 0) {
    while (Serial1.available() > 0) {
      char userInput = Serial1.read();
      if (userInput >= '0' && userInput <= '9') {
        throttle = (userInput - '0') * 25;
      } else if (userInput == 'a') { //go left
        tAngleY = -15.0;
      } else if (userInput =='d') { //go right
        tAngleY = +15.0;
      } else if(userInput =='s') { //balance
        tAngleX = 0.0;
        tAngleY = 0.0;
        tAngleZ = 0.0;
      } else if(userInput =='w') { //go forward
        tAngleX = -15.0;
      } else if(userInput =='z') { //go back
        tAngleX = 15.0;
      } else if(userInput =='q') { //turn ccw
        tAngleZ = +15.0;
      } else if(userInput =='e') { //turn cw
        tAngleZ = -15.0;
      }


    }
  }
```

- 숫자 입력 → throttle 조절 (0~225 범위)
w/a/s/d → 목표 각도 변경 (앞/왼/정지/오른쪽)
q/e → 회전 (좌/우 회전)

```
  double speedA = throttle + BalY + BalX + BalZ;
  double speedB = throttle - BalY + BalX - BalZ;
  double speedC = throttle - BalY - BalX + BalZ;
  double speedD = throttle + BalY - BalX - BalZ;
```

- 4개의 모터 출력에 PID 제어값을 조합하여 분배.
constrain으로 0~250 범위 제한.
analogWrite로 PWM 출력 → 모터 속도 제어.

```
  int iSpeedA = constrain((int)speedA, 0, 250);
  int iSpeedB = constrain((int)speedB, 0, 250);
  int iSpeedC = constrain((int)speedC, 0, 250);
  int iSpeedD = constrain((int)speedD, 0, 250);

  // 모터 구동
  analogWrite(6, iSpeedA);
  analogWrite(10, iSpeedB);
  analogWrite(9, iSpeedC);
  analogWrite(5, iSpeedD);
}
```

앞선 코드들을 요약하자면

1. MPU-6050에서 각속도 읽기 → 보정 → 적분 → 각도 추정
2. 목표 각도와 현재 각도 오차 계산 → PID 계산
3. 사용자 입력에 따라 목표 각도/쓰로틀 변경
4. PID 값과 쓰로틀을 합산해 4개 모터 속도 계산 → PWM 출력

의 흐름으로 설명할 수 있다.

2. 실험 및 탐구 결과

실습 1

PWM 주기와 Duty cycle이 모터 동작에 주는 영향을 확인했다. 주기를 변경했을 때 주기가 길수록 모터가 부자연스러웠고 주기가 짧을수록 더 매끄럽게 돌아갔다. Dury cycle의 값을 변경했을 때 값이 더 클수록 모터의 회전력과 속도가 빨라지는 걸 볼 수 있었다. analogWrite 함수를 사용하면 코드가 단순해지고 다양한 Dutycycle 값을 쉽게 적용할 수 있었다.

실습 2

시리얼 모니터에서 출력된 데이터를 앞에서 배운 MSP 패킷 형식과 비교했다. 헤더, 데이터 크기, 메시지 타입, roll, pitch, yaw, throttle 등의 값들을 확인해 드론 조종 앱에서 보낸 신호가 아두이노를 통해 정상적으로 수신됨을 알 수 있었다.

MSP 메시지 중에서 시리얼 모니터에 roll, pitch, yaw, throttle의 값만 추출돼 표시되었음을 확인해 이 값을 드론에 전달해 드론 모터를 조종기를 통해 제어한다는 걸 알았다.

블루투스 모듈을 이용해 직접 throttle 값을 입력하였더니 0에서는 모터가 정지하고 1~4의 숫자를 입력하면 값이 점점 커질수록 모터의 출력도 점차 강해짐을 확인했다.

실습 3

시리얼 모니터에 나온 값은 자이로 센서 자체의 각속도 값이라 바로 제어에 활용할 수 없고 누적 오차가 발생해 자이로 오차 값의 평균을 측정된 각속도에서 빼주고 °/s 단위로 바꾼 뒤 Δt를 계산한 후 적분하여 각도를 추정하였다. 보정 후에는 회전 각도의 변화를 추적할 수 있었지만, 장시간 측정에서는 여전히 drift 현상이 발생해 완벽한 보정이 필요함을 확인했다.

실습 4

비례 제어를 관찰한 결과 드론을 기울이면 기울어진 방향 모터 출력이 증가해 균형을 맞추려는 반응이 일어났다. 드론의 회전 각도가 목표치에 도달해도 계속 진동하며 변화해 드론이 불안정하게 흔들림을 보고 안정적인 제어가 필요하다는 걸 확인했다.

비례 미분 제어에서는 비례 제어에 비해 모터 출력이 증가 후 점점 감소해 전보다 안정적이었지만 완전히 안정적이지는 않았다.

비례 적분 미분 제어에서는 기울어진 쪽 모터 출력이 점점 증가해 균형이 맞춰졌고 반대 방향으로 기울였을 때 출력이 감소해 앞선 두 과정보다 훨씬 안정적 제어가 가능함을 보았다.

드론 비행 실습

throttle 값의 변화를 주며 드론을 조종했을 때, 3.0의 값에서는 땅에서부터 5cm정도 상승한 상태를 유지했다. 드론을 조종하며 비행의 안정성이나 작동 상태를 확인하기 위해 3.6의 값을 입력했을 때는 드론이 점점 상승하였다. 드론을 처음 높이 비행시켰을 때 PID 계수를 Kp=1.0, Ki=1.0, Kd=1.0으로 설정했더니 드론이 뒤집히는 현상이 발생했다. 이후 PID 계수를 0.1 단위로 세밀하게 조절하며 실험한 결과 드론의 반응이 점차 안정화되었다. 특히 Kp와 Kd를 0.3~0.5 범위로 낮추고 Ki를 적절히 조절하자 드론이 기울어졌을 때 모터 출력이 과도하게 증가하지 않고 목표 각도로 부드럽게 복귀하였다.

3. 분석 및 토의

PWM 주기가 짧을수록 모터가 매끄럽게 회전하고, Duty cycle이 클수록 회전력과 속도가 증가함을 통해 드론 모터 제어에서 PWM 신호 조절이 얼마나 직관적이고 중요한지 알 수 있었다. 드론 모터 제어는 PWM 주기와 Duty cycle 조절만으로도 기본적인 회전력과 속도를 제어할 수 있으며, 이를 정확히 이해해야 안정적인 비행이 가능하다는 의미를 알 수 있었다.

시리얼 모니터를 통해 MSP 패킷을 분석하고, roll, pitch, yaw, throttle 신호가 드론으로 정상적으로 전달되는 것을 확인하였다. 이를 통해 자이로 센서 데이터를 단순히 읽는 것만으로는 제어가 어렵고, 센서 보정과 각도 추정이 필수적이라는 걸 알았다.

자이로 센서 값을 측정하고 보정하는 과정을 통해 드론의 회전 각도를 추적하였다. 원시 자이로 데이터는 바로 제어에 활용할 수 없고 시간이 지남에 따라 drift가 발생했다. 이를 보정하기위해 각속도 평균을 빼고 °/s 단위로 변환한 후 Δt를 계산하고 적분하여 각도를 추정하는 과정을 통해 센서 데이터를 직접 보정하고, 장시간 측정에서도 최대한 정확하게 회전 각도를 추적하는 방법을 확인했다.

PID 제어기의 적용과 조정 과정을 통해 드론이 가장 안정적이게 날 수 있는 상태를 찾았다. PID 계수를 Kp=1.0, Ki=1.0, Kd=1.0으로 설정하면 드론이 뒤집히는 현상은 모터 출력이 과도하게 반응하여 발생한 문제라고 할 수 있다. PID 계수를 0.1 단위로 조절해 드론의 반응이 점차 안정화되었는데, 이 과정에서 각각의 계수와 드론의 움직임을 보며 무엇이 문제인지 고려해보았다. 맨처음 드론이 불안정하게 날 때 Kp의 값을 조절해봤는데 Kp값이 작아지면 한쪽으로 기울게 되었다. 이는 Kp값이 크지 않아 기울어진 곳에 충분한 힘이 들어가지 않음으로써 기울어졌다고 해석할 수 있었다. 반대로 이 값이 커졌을 때는 드론이 기울어졌을 때 기울어진 쪽에 너무 큰 힘이 들어가 뒤집어졌다고 이해할 수 있었다.

Kd 값을 조절했을 때, 값을 전보다 작게 한 후에 비행을 하면 드론의 흔들림이 계속되었는데 이는 회전속도 상쇄 힘을 적게 주는 것과 같다는 의미라는 걸 알 수 있었다. 값이 크면 드론이 뒤집어졌는데, 이는 드론의 회전에 지나치게 민감하게 반응해서 드론이 뒤집어졌다고 볼 수 있었다. Ki 값이 상대적으로 안정적일 때보다 작을 때는 드론이 안정화된 상태로 가지 못하였다. 이는 Ki 값이 작아 누적 오차를 충분히 보정하지 못했기 때문에 드론이 목표 각도로 부드럽게 복귀하지 못했음을 의미한다. Ki 값을 너무 크게 설정하면 드론이 안정화된 상태로 너무 빨리 되돌아가서 흔들림이 발생했었다.

결국 PID 계수는 서로 상호작용하며, 한 계수만 조정하는 것보다 세 계수를 종합적으로 조정하여 드론 특성에 맞게 세밀하게 조정하는 게 안정적인 비행을 위해 중요함을 확인했다. 이를 바탕으로 드론 조종 실습에서 Kp와 Kd를 적절히 낮추고 Ki를 조정함으로써 드론의 과도한 반응과 진동을 최소화하고, 안정적인 드론 비행을 볼 수 있었다.

4. 결론 및 성찰

이번 활동을 통해 드론이 단순히 모터의 출력만으로 제어되는 것이 아니라, 센서 데이터를 기반으로 한 정밀한 피드백 제어가 필수적이라는 것을 깊이 깨달았다. 특히 자이로 센서를 사용하면서, 처음에는 센서 값이 시간이 지날수록 조금씩 틀어지는 '누적 오차(drift)' 현상이 발생해 원하는 제어가 어려웠다. 이를 보정하기 위해 약 1000회의 샘플 데이터를 수집해 평균 오차를 계산한 뒤, 이후 측정값에서 이를 실시간으로 빼 주는 보정 과정을 적용하였다. 그런 다음 단위를 °/s로 변환한 뒤 주기 계산을 통해 각속도를 적분해 회전 각도로 바꾸는 과정을 거쳤다. 이 과정을 통해 단순히 값만 읽는 것이 아니라 센서 데이터의 특성을 이해하고 다듬어야 제어의 성능을 향상시킬 수 있다는 것을 깨달았다.

드론 비행 실습 과정에서 Kp, Kd, Ki의 값을 각각 1.0으로 했을 때 드론이 계속해서 불안정적으로 흔들리며 날았다. 이를 보고 Kp와 Kd의 값이 커서 드론이 기울어짐에 예민하게 반응해 이러한 반응이 일어났다고 생각했다. 따라서 각각의 값을 0.1의 단위로 조정하면서 드론의 비행을 관찰했다. 결론적으로는 Kp, Kd를 각각 0.5와 0.7로 설정했을 때 드론이 처음보다 안정적이었지만 throttle을 3.6으로 입력하고 s를 입력했음에도 드론이 기울어진 상태로 비행했다. 원래라면 드론이 제자리에서 정지한 채로 위로 상승만 해야하는데 상승함과 동시에 기울어진 채로 옆으로 이동했다. 이를 해결하기 위해 Ki 값을 1.2정도로 약간 높였더니 드론의 기울어짐이 보정되어 드론이 기울어지지 않고 제자리에서 잘 비행할 수 있었다. 이 활동을 통해 PID 제어에서 각각의 요소가 비행 안정성에 어떤 영향을 미치는지를 실제 비행을 통해 체감할 수 있었다. 특히 Kp와 Kd는 드론의 기울기에 민감하게 반응해 드론이 흔들리거나 불안정하게 비행할 수 있다는 것을 확인할 수 있었고, Ki 값을 적절히 조정해 기울어짐을 보정하고 안정적으로 드론이 상승하게 할 수 있어서 보람찼다. 이렇게 안정적인 상태가 되기까지 PID의 값을 여러 번 바꿔보고 특히 0.1의 단위로 조정해보는 게 힘들었지만 드론 제어에서 얼마나 세밀한 조정이 필요한지 이해할 수 있었다.

앞으로 확장할 수 있는 탐구 방향

1. 딥러닝 기반 센서를 활용한 노이즈 제거 / 예측 보정

드론의 자이로·가속도 센서는 흔들림이나 환경 요인 때문에 원래 값이 아닌 노이즈가 섞여 나온다. 따라서 이 값을 그대로 쓰게 된다면 드론의 움직임 계산이 불안정해진다. 이동평균이나 칼만 필터와 같이 단순히 값 평균을 내는 필터 대신에 딥러닝(LSTM, GRU 같은 시계열 신경망)을 사용한다. 이를 통해 센서가 앞으로 낼 값을 예측하고 실제 값과 비교하면서 노이즈를 제거한다. 그렇다면 기존 PID 입력값의 정확도를 크게 높여서 드론이 더 안정적으로 비행할 수 있게 된다. 예를 들어, 드론이 바람 때문에 센서가 순간적으로 +5° 기울어진 값으로 튀었을 때, LSTM이 실제로는 2° 정도일 것을 추정해 PID에 넣어준다면 불필요한 흔들림 방지할 수 있다.

2.AI 기반 비행 제어 자동 학습

드론은 기본적으로 PID 제어로 안정성을 유지하지만 외부 조건이 달라지면 사람이 직접 Kp, Ki, Kd 값을 조정해야 한다. 이 과정은 환경이 계속 바뀌는 실제 비행에서 한계가 있다. 이를 개선하기 위해 인공지능을 활용하여 드론이 스스로 최적의 제어 방식을 학습하도록 만들 수 있다. PID 자동 튜닝을 통해 AI 기법으로 드론이 비행 중 오차, 진동 등을 스스로 측정하고 그에 따라 PID 값을 제어한다. 시뮬레이터에서 다양한 조건을 주고 드론이 안정적인 모터 출력 패턴을 학습한 후 실제 드론에 적용한다. 환경 변화에 유연하게 적응할 수 있고 사람이 직접 수치를 맞추지 않아도 되므로 조정이 간단해진다. 드론이 목표 고도를 유지하려 할 때 갑자기 바람이 불면 기존 PID 제어는 같은 Kp, Ki, Kd 값으로 계속 보정해 진동이 커질 수 있다. 하지만 AI 기반 제어에서는 드론이 즉시 새로운 환경에 맞게 파라미터를 조정하거나, 강화학습을 통해 이미 학습한 규칙을 적용해 바람에도 안정적으로 고도를 유지할 수 있다.