|  |
| --- |
| **2024 ALTIS SW 개인 활동 보고서** |

|  |  |
| --- | --- |
| **활동 개요** | 1차 SW팀 교육, 가속도 자이로 값으로 각도 추정하는 방법(과제) |
| **일시** | 2024년 04월 11일 19시 |
| **작성자** | 양주호 |
| **활동 내용** | ● 1차 SW팀 교육  1. MPU6050  - MPU6050를 구성하는 주요 센서로는 가속도센서, 자이로센서가 있다. 이는 자세 제어, 각속도, 가속도 측정, 자세 추정 등 다양한 제어 및 감지 시스템에 활용된다.  - 예제를 이용한 RAW데이터 추출 코드  #include <Adafruit\_MPU6050.h>  #include <Adafruit\_Sensor.h>  #include <Wire.h>  Adafruit\_MPU6050 mpu; // MPU6050 객체 생성  void setup(void) {  Serial.begin(115200); // 시리얼 통신 시작  while (!Serial)  delay(10); // 시리얼 통신을 기다립니다.  Serial.println("Adafruit MPU6050 test!"); // 초기화 메시지 출력  // MPU6050 초기화  if (!mpu.begin()) { // MPU6050 초기화에 실패하면  Serial.println("Failed to find MPU6050 chip"); // 실패 메시지 출력  while (1) {  delay(10); // 멈춤  }  }  Serial.println("MPU6050 Found!"); // 성공 메시지 출력  // 가속도계 범위 설정  mpu.setAccelerometerRange(MPU6050\_RANGE\_8\_G);  Serial.print("Accelerometer range set to: ");  switch (mpu.getAccelerometerRange()) {  case MPU6050\_RANGE\_2\_G:  Serial.println("+-2G");  break;  case MPU6050\_RANGE\_4\_G:  Serial.println("+-4G");  break;  case MPU6050\_RANGE\_8\_G:  Serial.println("+-8G");  break;  case MPU6050\_RANGE\_16\_G:  Serial.println("+-16G");  break;  }  // 자이로스코프 범위 설정  mpu.setGyroRange(MPU6050\_RANGE\_500\_DEG);  Serial.print("Gyro range set to: ");  switch (mpu.getGyroRange()) {  case MPU6050\_RANGE\_250\_DEG:  Serial.println("+- 250 deg/s");  break;  case MPU6050\_RANGE\_500\_DEG:  Serial.println("+- 500 deg/s");  break;  case MPU6050\_RANGE\_1000\_DEG:  Serial.println("+- 1000 deg/s");  break;  case MPU6050\_RANGE\_2000\_DEG:  Serial.println("+- 2000 deg/s");  break;  }  // 필터 대역폭 설정  mpu.setFilterBandwidth(MPU6050\_BAND\_21\_HZ);  Serial.print("Filter bandwidth set to: ");  switch (mpu.getFilterBandwidth()) {  case MPU6050\_BAND\_260\_HZ:  Serial.println("260 Hz");  break;  case MPU6050\_BAND\_184\_HZ:  Serial.println("184 Hz");  break;  case MPU6050\_BAND\_94\_HZ:  Serial.println("94 Hz");  break;  case MPU6050\_BAND\_44\_HZ:  Serial.println("44 Hz");  break;  case MPU6050\_BAND\_21\_HZ:  Serial.println("21 Hz");  break;  case MPU6050\_BAND\_10\_HZ:  Serial.println("10 Hz");  break;  case MPU6050\_BAND\_5\_HZ:  Serial.println("5 Hz");  break;  }  Serial.println("");  delay(100);  }  void loop() {  /\* 새로운 센서 이벤트 및 읽기값을 얻어옵니다. \*/  sensors\_event\_t a, g, temp;  mpu.getEvent(&a, &g, &temp);  /\* 값을 출력합니다. \*/  Serial.print("Acceleration X: ");  Serial.print(a.acceleration.x);  Serial.print(", Y: ");  Serial.print(a.acceleration.y);  Serial.print(", Z: ");  Serial.print(a.acceleration.z);  Serial.println(" m/s^2");  Serial.print("Rotation X: ");  Serial.print(g.gyro.x);  Serial.print(", Y: ");  Serial.print(g.gyro.y);  Serial.print(", Z: ");  Serial.print(g.gyro.z);  Serial.println(" rad/s");  Serial.print("Temperature: ");  Serial.print(temp.temperature);  Serial.println(" degC");  Serial.println("");  delay(500);  }  2. BMP280  - 대기압과 온도를 측정하는 디지털 압력 센서이다. 이때, ‘bmp.begin()’의 주소는 센서에 연결된 SDO핀의 연결 상태에 따라 결정되는데 GND핀에 연결되어 있기 때문에 ‘0x76’이 주소값이 된다.  - 예제를 이용한 RAW데이터 추출 코드  #include <Wire.h>  #include <SPI.h>  #include <Adafruit\_BMP280.h>  #define BMP\_SCK (13)  #define BMP\_MISO (12)  #define BMP\_MOSI (11)  #define BMP\_CS (10)  Adafruit\_BMP280 bmp; // I2C  //Adafruit\_BMP280 bmp(BMP\_CS); // 하드웨어 SPI  //Adafruit\_BMP280 bmp(BMP\_CS, BMP\_MOSI, BMP\_MISO, BMP\_SCK);  void setup() {  Serial.begin(9600); // 시리얼 통신 시작  while ( !Serial ) delay(100); // 네이티브 USB 기다림  Serial.println(F("BMP280 test")); // 초기화 메시지 출력  unsigned status;  status = bmp.begin(0x76); // I2C 주소 0x76으로 BMP280 초기화  if (!status) { // 초기화 실패 시  Serial.println(F("Could not find a valid BMP280 sensor, check wiring or "  "try a different address!"));  Serial.print("SensorID was: 0x"); Serial.println(bmp.sensorID(),16);  Serial.print(" ID of 0xFF probably means a bad address, a BMP 180 or BMP 085\n");  Serial.print(" ID of 0x56-0x58 represents a BMP 280,\n");  Serial.print(" ID of 0x60 represents a BME 280.\n");  Serial.print(" ID of 0x61 represents a BME 680.\n");  while (1) delay(10); // 멈춤  }  /\* 데이터시트에서의 기본 설정값. \*/  bmp.setSampling(Adafruit\_BMP280::MODE\_NORMAL, /\* 동작 모드. \*/  Adafruit\_BMP280::SAMPLING\_X2, /\* 온도 샘플링 \*/  Adafruit\_BMP280::SAMPLING\_X16, /\* 압력 샘플링 \*/  Adafruit\_BMP280::FILTER\_X16, /\* 필터링. \*/  Adafruit\_BMP280::STANDBY\_MS\_500); /\* 대기 시간. \*/  }  void loop() {  Serial.print(F("Temperature = "));  Serial.print(bmp.readTemperature()); // 온도 측정 및 출력  Serial.println(" \*C");  Serial.print(F("Pressure = "));  Serial.print(bmp.readPressure()); // 압력 측정 및 출력  Serial.println(" Pa");  Serial.print(F("Approx altitude = "));  Serial.print(bmp.readAltitude(1013.25)); /\* Adjusted to local forecast! \*/ // 고도 측정 및 출력  Serial.println(" m");  Serial.println();  delay(2000); // 2초 대기  }  3. HC-12  - 무선 시리얼 통신 모듈이다. 433MHz 또는 868/915MHz 주파수 대역을 지원하고, 수십 미터에서 몇 킬로미터까지의 통신 범위를 제공한다.  - 예제를 이용한 RAW데이터 추출 코드  #define RXD2 16 //(RX2) - RX2 핀의 핀 번호 정의  #define TXD2 17 //(TX2) - TX2 핀의 핀 번호 정의  #define HC12 Serial2 // ESP32의 하드웨어 시리얼 2를 HC12로 정의  void setup()  {  pinMode(5, OUTPUT); // 5번 핀을 출력으로 설정  digitalWrite(5, LOW); // 기본적으로 HIGH이지만, 설정을 위해 LOW로 설정  Serial.begin(115200); // 컴퓨터로의 시리얼 통신을 위한 시리얼 포트 설정  HC12.begin(9600, SERIAL\_8N1, RXD2, TXD2); // HC12로의 시리얼 통신 설정, 9600 보율, 8N1 포맷, RX2 핀과 TX2 핀 사용  }  void loop()  {  while (HC12.available())  {  // 만약 HC-12로부터 데이터가 도착했다면  Serial.write(HC12.read()); // 그 데이터를 시리얼 모니터로 보냅니다.  }  while (Serial.available())  {  // 만약 시리얼 모니터로부터 데이터가 도착했다면  HC12.write(Serial.read()); // 그 데이터를 HC-12로 보냅니다.  }  }  ● (과제) 가속도 자이로 값으로 각도 추정하는 방법  1. MPU-6050를 이용한 각도추정  ○ 가속도센서  - 가속도센서는 3축의 중력가속도를 포함하여 총 가속도를 측정한다. 중력가속도는 일정한 크기와 방향을 가지기 때문에 센서가 각 축에 수직 방향일 때 중력 가속도 성분이 최대가 되고, 물체가 수직에서 벗어나면 중력 가속도의 성분이 변화한다. 이러한 물체의 움직임에 따른 중력 가속도 변화량을 측정하여 각도를 추정한다. 가속도센서를 이용해 동적 물체에 경우 가속도와 물체의 중력이 함께 측정되므로 정확한 추정이 어렵기 때문에 정적인 상태일수록 정확한 각도의 데이터를 추정할 수 있음.  - 코드 (롤,피치만 구하는 예제) #include<Wire.h>  const int MPU\_ADDR = 0x68; // I2C통신을 위한 MPU6050의 주소  int16\_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ; // 가속도(Acceleration)와 자이로(Gyro)  double angleAcX;  double angleAcY;  const double RADIAN\_TO\_DEGREE = 180 / 3.14159;  void setup() {  initSensor();  Serial.begin(9600);  delay(200);  }  void loop() {  getAngleXY();  Serial.print("Angle x : ");  Serial.print(angleAcX);  Serial.print("\t\t Angle y : ");  Serial.println(angleAcY);  delay(20);  }  double getAngleXY() {  getData();  // 삼각함수를 이용한 롤(Roll)의 각도 구하기  angleAcX = atan(AcY / sqrt(pow(AcX, 2) + pow(AcZ, 2)));  angleAcX \*= RADIAN\_TO\_DEGREE;  // 삼각함수를 이용한 피치(Pitch)의 각도 구하기  angleAcY = atan(-AcX / sqrt(pow(AcY, 2) + pow(AcZ, 2)));  angleAcY \*= RADIAN\_TO\_DEGREE;  }  void initSensor() {  Wire.begin();  Wire.beginTransmission(MPU\_ADDR); // I2C 통신용 어드레스(주소)  Wire.write(0x6B); // MPU6050과 통신을 시작하기 위해서는 0x6B번지에  Wire.write(0);  Wire.endTransmission(true);  }  void getData() {  Wire.beginTransmission(MPU\_ADDR);  Wire.write(0x3B); // AcX 레지스터 위치(주소)를 지칭합니다  Wire.endTransmission(false);  Wire.requestFrom(MPU\_ADDR, 14, true); // AcX 주소 이후의 14byte의 데이터를 요청  AcX = Wire.read() << 8 | Wire.read(); //두 개의 나뉘어진 바이트를 하나로 이어 붙여서 각 변수에 저장  AcY = Wire.read() << 8 | Wire.read();  AcZ = Wire.read() << 8 | Wire.read();  Tmp = Wire.read() << 8 | Wire.read();  GyX = Wire.read() << 8 | Wire.read();  GyY = Wire.read() << 8 | Wire.read();  GyZ = Wire.read() << 8 | Wire.read();  }  ○ 자이로센서  - 자이로센서는 3축의 각속도를 이용하여 각도를 추정한다. 자이로센서로 각 축의 각속도를 측정하고, 측정한 각속도는 시간에 대한 데이터이기 때문에 시간에 대해 적분하면 각 축의 회전 각도 변화량이 나온다. 3축의 각속도를 적분한 값을 종합하여 현재 물체의 각도를 추정한다. 다만 시간에 대해 적분을 하기 때문에 각도 데이터를 쌓는 과정에서 오차가 발생할 때 그 다음 각도 계산 과정에서 그 오차를 포함하여 계산하기 때문에 오차가 누적될 수 있다.  - 코드  #include<Wire.h>  const int MPU\_ADDR = 0x68; // I2C통신을 위한 MPU6050의 주소  int16\_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ; // 가속도(Acceleration)와 자이로(Gyro)  double angleAcX, angleAcY, angleAcZ;  double angleGyX, angleGyY, angleGyZ;  const double RADIAN\_TO\_DEGREE = 180 / 3.14159;  const double DEG\_PER\_SEC = 32767 / 250; // 1초에 회전하는 각도  // GyX, GyY, GyZ 값의 범위 : -32768 ~ +32767 (16비트 정수범위)  unsigned long now = 0; // 현재 시간 저장용 변수  unsigned long past = 0; // 이전 시간 저장용 변수  double dt = 0; // 한 사이클 동안 걸린 시간 변수  double averAcX, averAcY, averAcZ;  double averGyX, averGyY, averGyZ;  void setup() {  initSensor();  Serial.begin(115200);  caliSensor(); // 초기 센서 캘리브레이션 함수 호출  past = millis(); // past에 현재 시간 저장  }  void loop() {  getData();  getDT();  angleGyX += ((GyX - averGyX) / DEG\_PER\_SEC) \* dt;  angleGyY += ((GyY - averGyY) / DEG\_PER\_SEC) \* dt;  angleGyZ += ((GyZ - averGyZ) / DEG\_PER\_SEC) \* dt;    Serial.print("Angle Gyro X:");  Serial.print(angleGyX);  Serial.print("\t\t Angle Gyro y:");  Serial.print(angleGyY);  Serial.print("\t\t Angle Gyro Z:");  Serial.println(angleGyZ);  delay(20);  }  void initSensor() {  Wire.begin();  Wire.beginTransmission(MPU\_ADDR); // I2C 통신용 어드레스(주소)  Wire.write(0x6B); // MPU6050과 통신을 시작하기 위해서는 0x6B번지에  Wire.write(0);  Wire.endTransmission(true);  }  void getData() {  Wire.beginTransmission(MPU\_ADDR);  Wire.write(0x3B); // AcX 레지스터 위치(주소)를 지칭합니다  Wire.endTransmission(false);  Wire.requestFrom(MPU\_ADDR, 14, true); // AcX 주소 이후의 14byte의 데이터를 요청  AcX = Wire.read() << 8 | Wire.read(); //두 개의 나뉘어진 바이트를 하나로 이어 붙여서 각 변수에 저장  AcY = Wire.read() << 8 | Wire.read();  AcZ = Wire.read() << 8 | Wire.read();  Tmp = Wire.read() << 8 | Wire.read();  GyX = Wire.read() << 8 | Wire.read();  GyY = Wire.read() << 8 | Wire.read();  GyZ = Wire.read() << 8 | Wire.read();  }  // loop 한 사이클동안 걸리는 시간을 알기위한 함수  void getDT() {  now = millis();  dt = (now - past) / 1000.0;  past = now;  }  // 센서의 초기값을 10회 정도 평균값으로 구하여 저장하는 함수  void caliSensor() {  double sumAcX = 0 , sumAcY = 0, sumAcZ = 0;  double sumGyX = 0 , sumGyY = 0, sumGyZ = 0;  getData();  for (int i=0;i<10;i++) {  getData();  sumAcX+=AcX; sumAcY+=AcY; sumAcZ+=AcZ;  sumGyX+=GyX; sumGyY+=GyY; sumGyZ+=GyZ;  delay(50);  }  averAcX=sumAcX/10; averAcY=sumAcY/10; averAcZ=sumAcY/10;  averGyX=sumGyX/10; averGyY=sumGyY/10; averGyZ=sumGyZ/10;  }  ○ 필터링 (상보필터)  - 가속도센서는 센서 특성상 고주파 영역에서 노이즈가 많이 발생하고 자이로센서는 저주파 영역에 대한 데이터의 정확도가 낮다. 상보필터는 가속도센서의 저주파 영역, 자이로센서의 고주파 영역에서의 장점을 활용하는 필터이고 각 센서로부터 얻은 데이터에 가중치를 적용해 보정하는 원리이다. - 코드  #include<Wire.h>  const int MPU\_ADDR = 0x68; // I2C통신을 위한 MPU6050의 주소  int16\_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ; // 가속도(Acceleration)와 자이로(Gyro)  double angleAcX, angleAcY, angleAcZ;  double angleGyX, angleGyY, angleGyZ;  double angleFiX, angleFiY, angleFiZ;  const double RADIAN\_TO\_DEGREE = 180 / 3.14159;  const double DEG\_PER\_SEC = 32767 / 250; // 1초에 회전하는 각도  const double ALPHA = 1 / (1 + 0.04);  // GyX, GyY, GyZ 값의 범위 : -32768 ~ +32767 (16비트 정수범위)  unsigned long now = 0; // 현재 시간 저장용 변수  unsigned long past = 0; // 이전 시간 저장용 변수  double dt = 0; // 한 사이클 동안 걸린 시간 변수  double averAcX, averAcY, averAcZ;  double averGyX, averGyY, averGyZ;  void setup() {  initSensor();  Serial.begin(115200);  caliSensor(); // 초기 센서 캘리브레이션 함수 호출  past = millis(); // past에 현재 시간 저장  }  void loop() {  getData();  getDT();  angleAcX = atan(AcY / sqrt(pow(AcX, 2) + pow(AcZ, 2)));  angleAcX \*= RADIAN\_TO\_DEGREE;  angleAcY = atan(-AcX / sqrt(pow(AcY, 2) + pow(AcZ, 2)));  angleAcY \*= RADIAN\_TO\_DEGREE;  // 가속도 센서로는 Z축 회전각 계산 불가함.    // 가속도 현재 값에서 초기평균값을 빼서 센서값에 대한 보정  angleGyX += ((GyX - averGyX) / DEG\_PER\_SEC) \* dt; //각속도로 변환  angleGyY += ((GyY - averGyY) / DEG\_PER\_SEC) \* dt;  angleGyZ += ((GyZ - averGyZ) / DEG\_PER\_SEC) \* dt;  // 상보필터 처리를 위한 임시각도 저장  double angleTmpX = angleFiX + angleGyX \* dt;  double angleTmpY = angleFiY + angleGyY \* dt;  double angleTmpZ = angleFiZ + angleGyZ \* dt;  // (상보필터 값 처리) 임시 각도에 0.96가속도 센서로 얻어진 각도 0.04의 비중을 두어 현재 각도를 구함.  angleFiX = ALPHA \* angleTmpX + (1.0 - ALPHA) \* angleAcX;  angleFiY = ALPHA \* angleTmpY + (1.0 - ALPHA) \* angleAcY;  angleFiZ = angleGyZ; // Z축은 자이로 센서만을 이용하열 구함.  Serial.print("AngleAcX:");  Serial.print(angleAcX);  Serial.print("\t FilteredX:");  Serial.print(angleFiX);  Serial.print("\t AngleAcY:");  Serial.print(angleAcY);  Serial.print("\t FilteredY:");  Serial.println(angleFiY);  Serial.print("\t AngleGyZ:");  Serial.print(angleGyZ);  Serial.print("\t FilteredZ:");  Serial.println(angleFiZ);  // Serial.print("Angle Gyro X:");  // Serial.print(angleGyX);  // Serial.print("\t\t Angle Gyro y:");  // Serial.print(angleGyY);  // Serial.print("\t\t Angle Gyro Z:");  // Serial.println(angleGyZ);  // delay(20);  }  void initSensor() {  Wire.begin();  Wire.beginTransmission(MPU\_ADDR); // I2C 통신용 어드레스(주소)  Wire.write(0x6B); // MPU6050과 통신을 시작하기 위해서는 0x6B번지에  Wire.write(0);  Wire.endTransmission(true);  }  void getData() {  Wire.beginTransmission(MPU\_ADDR);  Wire.write(0x3B); // AcX 레지스터 위치(주소)를 지칭합니다  Wire.endTransmission(false);  Wire.requestFrom(MPU\_ADDR, 14, true); // AcX 주소 이후의 14byte의 데이터를 요청  AcX = Wire.read() << 8 | Wire.read(); //두 개의 나뉘어진 바이트를 하나로 이어 붙여서 각 변수에 저장  AcY = Wire.read() << 8 | Wire.read();  AcZ = Wire.read() << 8 | Wire.read();  Tmp = Wire.read() << 8 | Wire.read();  GyX = Wire.read() << 8 | Wire.read();  GyY = Wire.read() << 8 | Wire.read();  GyZ = Wire.read() << 8 | Wire.read();  }  // loop 한 사이클동안 걸리는 시간을 알기위한 함수  void getDT() {  now = millis();  dt = (now - past) / 1000.0;  past = now;  }  // 센서의 초기값을 10회 정도 평균값으로 구하여 저장하는 함수  void caliSensor() {  double sumAcX = 0 , sumAcY = 0, sumAcZ = 0;  double sumGyX = 0 , sumGyY = 0, sumGyZ = 0;  getData();  for (int i=0;i<10;i++) {  getData();  sumAcX+=AcX; sumAcY+=AcY; sumAcZ+=AcZ;  sumGyX+=GyX; sumGyY+=GyY; sumGyZ+=GyZ;  delay(50);  }  averAcX=sumAcX/10; averAcY=sumAcY/10; averAcZ=sumAcY/10;  averGyX=sumGyX/10; averGyY=sumGyY/10; averGyZ=sumGyZ/10;  }  ○ 칼만필터  - 칼만 필터는 시스템의 상태를 예측하고, 실제 측정값과 예측값을 사용하여 노이즈를 줄이는 데 사용된다. 시스템의 동작을 모델링하여 동적인 예측 모델을 사용하는데, 이 예측 모델로 시스템의 다음 상태를 예측할 수 있고 이를 통해 현재 상태를 추정한다.  - 따라서 칼만필터의 핵심은 예측-보정단계로 이전 데이터로 현재를 예측하고 현재 데이터를 예측한 데이터로 보정하는 것이다.  - 과정  1. 시스템 모델링: 먼저 센서의 동작을 수학적으로 모델링해야 한다. 가장 기본적인 예시로는 자유낙하하는 물체가 있는데 위치, 속도, 시간 변수를 x,v,t라고 하면  x[k+1] = =x[k]+v[k]⋅Δt  v[k+1] = v[k]−g⋅Δt  라고 할 수 있고 이를 알고리즘으로 표현하는 것까지 시스템 모델링 과정이다.  2. 측정값 수집: 아두이노와 같은 마이크로 콘트롤러를 이용해 센서로부터 측정값을 읽는다.  3. 칼만 필터 구현: 구현된 칼만 필터 알고리즘을 사용하여 예측-보정 과정을 재귀적으로 측정값을 필터링한다.  - 코드(시스템 모델링 예시)  const float g = 9.81; // 중력 가속도 (m/s^2)  const float dt = 0.1; // 측정 주기 (s)  float x = 0; // 위치 (m)  float v = 0; // 속도 (m/s)  void setup() {  Serial.begin(9600);  }  void loop() {  // 시스템 모델링 (자유 낙하체 운동 방정식)  x = x + v \* dt;  v = v - g \* dt;  // 결과 출력  Serial.print("Position: ");  Serial.print(x);  Serial.print(" m, Velocity: ");  Serial.print(v);  Serial.println(" m/s");  delay(1000); // 1초 지연  }  - 칼만 필터 코드 예시  #include <Wire.h>  #define mpu\_add 0x68 // mpu6050 address  class kalman {  public:  double getkalman(double acc, double gyro, double dt) {  // 프로젝션 상태 갱신  angle += dt \* (gyro - bias);  // 프로젝션 오차 공분산 갱신  P[0][0] += dt \* (dt \* P[1][1] - P[0][1] - P[1][0] + Q\_angle);  P[0][1] -= dt \* P[1][1];  P[1][0] -= dt \* P[1][1];  P[1][1] += Q\_gyro \* dt;  // 칼만 이득 계산  double S = P[0][0] + R\_measure;  K[0] = P[0][0] / S;  K[1] = P[1][0] / S;  // 측정값을 이용하여 예측 갱신  double y = acc - angle;  angle += K[0] \* y;  bias += K[1] \* y;  // 오차 공분산 갱신  double P\_temp[2] = {P[0][0], P[0][1]};  P[0][0] -= K[0] \* P\_temp[0];  P[0][1] -= K[0] \* P\_temp[1];  P[1][0] -= K[1] \* P\_temp[0];  P[1][1] -= K[1] \* P\_temp[1];  return angle;  };  // 칼만 필터 초기화  void init(double angle, double gyro, double measure) {  Q\_angle = angle;  Q\_gyro = gyro;  R\_measure = measure;  angle = 0;  bias = 0;  P[0][0] = 0;  P[0][1] = 0;  P[1][0] = 0;  P[1][1] = 0;  };  // 변수값 반환  double getvar(int num) {  switch (num) {  case 0:  return Q\_angle;  break;  case 1:  return Q\_gyro;  break;  case 2:  return R\_measure;  break;  }  };  private:  double Q\_angle, Q\_gyro, R\_measure;  double angle, bias;  double P[2][2], K[2];  };  kalman kal;  long ac\_x, ac\_y, ac\_z, gy\_x, gy\_y, gy\_z;  double deg, dgy\_y;  double dt;  uint32\_t pasttime;  void setup() {  Serial.begin(9600);  Wire.begin();  Wire.beginTransmission(mpu\_add);  Wire.write(0x6B);  Wire.write(0);  Wire.endTransmission(true);  // 칼만 필터 초기화  kal.init(0.001, 0.003, 0.03);  Serial.println();  Serial.print("parameter");  Serial.print("\t");  Serial.print(kal.getvar(0), 4);  Serial.print("\t");  Serial.print(kal.getvar(1), 4);  Serial.print("\t");  Serial.println(kal.getvar(2), 4);  }  void loop() {  Wire.beginTransmission(mpu\_add);  Wire.write(0x3B);  Wire.endTransmission(false);  Wire.requestFrom(mpu\_add, 6, true);  ac\_x = Wire.read() << 8 | Wire.read();  ac\_y = Wire.read() << 8 | Wire.read();  ac\_z = Wire.read() << 8 | Wire.read();  Wire.beginTransmission(mpu\_add);  Wire.write(0x43);  Wire.endTransmission(false);  Wire.requestFrom(mpu\_add, 6, true);  gy\_x = Wire.read() << 8 | Wire.read();  gy\_y = Wire.read() << 8 | Wire.read();  gy\_z = Wire.read() << 8 | Wire.read();  // 가속도 데이터를 각도 데이터로 변환  deg = atan2(ac\_x, ac\_z) \* 180 / PI;  // 자이로 출력을 이해 가능한 값으로 변환  dgy\_y = gy\_y / 131.;  // 이전 측정시간으로부터 경과한 시간을 계산  dt = (double)(micros() - pasttime) / 1000000;  pasttime = micros();  // 칼만 필터를 이용하여 각도 측정  double val = kal.getkalman(deg, dgy\_y, dt);  Serial.print("kalman degree");  Serial.print("\t");  Serial.println(val);  } |
| **활동 사진** | 1. 롤, 피치의 각도를 구하기 위한 삼각함수 수식      1. 가속도 센서를 이용한 각도 계산 결과      1. 자이로 센서를 이용한 각도 계산 결과      1. 상보 필터를 적용한 각도 계산 결과 |
| **활동 결과** | (활동 사진에 첨부한 계산 결과 참조) |
| **참고 문헌** | [【 아두이노 완공 #29 】 MPU6050 자이로 가속도 센서 , 바로 이 영상으로 시작하세요~! (youtube.com)](https://www.youtube.com/watch?v=BTpwAFfeg0I) |

2024년 4월 13일

작성자 : 양주호 (인)