*장현주\_2023011923*

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

|  |  |
| --- | --- |
| **활동 개요** | 로켓의 각도 추정 방법에 대하여 조사(MPU6050) |
| **일시** | 2024년 4월 12일 00시 |
| **작성자** | 장현주 |
| **활동 내용** | *※ 어떤 활동을 진행하였는지 자세히 쓰기*  - MPU 6050 = IMU역할을 수행하는 초소형 반도체 칩이다. (IMU는 관성측정장치)  - IMU는 물체에 작용하는 가속력과 진동력, 충격력 등 동적 힘이 발생했을 때, 움직임의 변화에 따른 **가속도의 변화**를 순간적으로 감지한다.  센서에서 얻어지는 출력값인 가속도를 적분하여 물체의 진행 방향에 대한 속도를 계산할 수 있고, 이를 가공하면 물체의 위치를 알아낼 수 있다. 짧은 기간에 대한 오차는 작으나, 센서에서 얻은 가속도 값을 적분하므로 시간에 따라 **오차가 누적**되는 단점이 있다. ( **GPS와 결합하여 보정 되기도 한다)**  - 칼만 필터는 상태 예측과 측정 업데이트를 반복적으로 수행하며 자세를 계산하는 과정이다.  <칼만필터 단계>  측정값을 Z, 예측값을 X 으로 사용  칼만 Gain을 계산하게 되어 추정값 계산에서 측정값과 예측값의 비율을 결정  >만일 칼만 Gain이 크다면 측정값이 큰 비율로 들어가게 되고, 칼만 Gain이 작다면 예측값이 큰 비율로 들어가게 됨  *※ 코드가 있다면 어떤 코드인지 + 코드에 주석 달기*   * MPU첫번째 – 각도 값 계산 및 진동제거(인터넷)   #include <Wire.h>  void setup() {    Serial.begin(115200); // Serial 통신 초기화    Wire.begin(); // Wire 라이브러리 초기화  // 전원 관리 설정    Wire.beginTransmission(0x68); // MPU6050의 주소    Wire.write(107); // 전원 관리 레지스터 주소    Wire.write(0); // 전원 관리 레지스터 값 설정 (0)    Wire.endTransmission();    // 레지스터 26 설정    for(uint8\_t i = 2; i <= 7; i++)    {      Wire.beginTransmission(0x68); // MPU6050의 주소      Wire.write(26); // 레지스터 26 주소      Wire.write(i << 3 | 0x03); // 레지스터 26 값 설정 (비트 시프트 연산)      Wire.endTransmission();  }    // 레지스터 27 설정    Wire.beginTransmission(0x68); // MPU6050의 주소    Wire.write(27); // 레지스터 27 주소    Wire.write(3 << 3); // 레지스터 27 값 설정 (비트 시프트 연산)  Wire.endTransmission();    // 레지스터 28 설정    Wire.beginTransmission(0x68); // MPU6050의 주소    Wire.write(28); // 레지스터 28 주소    Wire.write(0); // 레지스터 28 값 설정 (0)    Wire.endTransmission();  }  int16\_t offset[3] = {-22, 15, -4};  void loop() {    uint8\_t i;  static int16\_t acc\_raw[3]={0,}, gyro\_raw[3]={0,};   // 가속도 값 얻기    Wire.beginTransmission(0x68); // MPU6050의 주소    Wire.write(59); // 가속도 레지스터 시작 주소    Wire.endTransmission();    Wire.requestFrom(0x68, 6); // 6바이트 읽기 요청    for(i = 0; i < 3; i++) acc\_raw[i] = (Wire.read() << 8) | Wire.read();// 가속도 값 읽기    // 자이로스코프 값 얻기    Wire.beginTransmission(0x68); // MPU6050의 주소    Wire.write(67); // 자이로스코프 레지스터 시작 주소    Wire.endTransmission();    Wire.requestFrom(0x68, 6); // 6바이트 읽기 요청    for(i = 0; i < 3; i++)      gyro\_raw[i] = gyro\_raw[i] \* 0.8 + 0.2 \* (((Wire.read() << 8) | Wire.read()) - offset[i]); //자이로스코프 값 읽기 및 보정  // 시간 간격 얻기    static unsigned long p = 0;    unsigned long c = micros();    float dt = (c - p) \* 0.000001F; // 시간 간격 계산  p = c;   // 자이로 각속도 계산    float gyro\_rate[3];    for(i = 0; i < 3; i++) gyro\_rate[i] = gyro\_raw[i] / 16.4 \* dt; // 자이로 각속도 계산    // 각도 계산    static float angle[3]={0,}, vec;    vec = sqrt(pow(acc\_raw[0], 2) + pow(acc\_raw[2], 2)); // 가속도 벡터 계산    angle[0] = (angle[0] + gyro\_rate[0]) \* 0.98      + atan2(acc\_raw[1], vec) \* RAD\_TO\_DEG \* 0.02; // X축 각도 계산    vec = sqrt(pow(acc\_raw[1], 2) + pow(acc\_raw[2], 2)); // 가속도 벡터 계산    angle[1] = (angle[1] - gyro\_rate[1]) \* 0.98      + atan2(acc\_raw[0], vec) \* RAD\_TO\_DEG \* 0.02; // Y축 각도 계산    // 시리얼 통신으로 출력    angle[2] += gyro\_rate[2]; // Z축 각도 계산    char str[50], a1[10], a2[10], a3[10];    dtostrf(angle[0], 4, 3, a1); // 각도를 문자열로 변환    dtostrf(angle[1], 4, 3, a2); // 각도를 문자열로 변환    dtostrf(angle[2], 4, 3, a3); // 각도를 문자열로 변환    sprintf(str, "X:%s Y:%s Z:%s", a1, a2, a3); // 포맷  문자열 생성    Serial.println(str); // 시리얼 모니터로 출력  }   * MPU두번째 – 센서로 값을 읽어오고 시리얼 모니터에 출력(인터넷)/ I2C통신 가능   #include<Wire.h>  const int MPU=0x68; //MPU 6050 의 I2C 기본 주소  int16\_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;  void setup(){  Wire.begin(); //Wire 라이브러리 초기화  Wire.beginTransmission(MPU); //MPU로 데이터 전송 시작  Wire.write(0x6B); // PWR\_MGMT\_1 register  Wire.write(0); //MPU-6050 시작 모드로  Wire.endTransmission(true);  Serial.begin(9600);  }  void loop(){  Wire.beginTransmission(MPU); //데이터 전송시작  Wire.write(0x3B); // register 0x3B (ACCEL\_XOUT\_H), 큐에 데이터 기록  Wire.endTransmission(false); //연결유지  Wire.requestFrom(MPU,14,true); //MPU에 데이터 요청  //데이터 한 바이트 씩 읽어서 반환  AcX=Wire.read()<<8|Wire.read(); // 0x3B (ACCEL\_XOUT\_H) & 0x3C (ACCEL\_XOUT\_L)  AcY=Wire.read()<<8|Wire.read(); // 0x3D (ACCEL\_YOUT\_H) & 0x3E (ACCEL\_YOUT\_L)  AcZ=Wire.read()<<8|Wire.read(); // 0x3F (ACCEL\_ZOUT\_H) & 0x40 (ACCEL\_ZOUT\_L)  Tmp=Wire.read()<<8|Wire.read(); // 0x41 (TEMP\_OUT\_H) & 0x42 (TEMP\_OUT\_L)  GyX=Wire.read()<<8|Wire.read(); // 0x43 (GYRO\_XOUT\_H) & 0x44 (GYRO\_XOUT\_L)  GyY=Wire.read()<<8|Wire.read(); // 0x45 (GYRO\_YOUT\_H) & 0x46 (GYRO\_YOUT\_L)  GyZ=Wire.read()<<8|Wire.read(); // 0x47 (GYRO\_ZOUT\_H) & 0x48 (GYRO\_ZOUT\_L)    //시리얼 모니터에 출력  Serial.print("AcX = "); Serial.print(AcX);  Serial.print(" | AcY = "); Serial.print(AcY);  Serial.print(" | AcZ = "); Serial.print(AcZ);  Serial.print(" | Tmp = "); Serial.print(Tmp/340.00+36.53);  Serial.print(" | GyX = "); Serial.print(GyX);  Serial.print(" | GyY = "); Serial.print(GyY);  Serial.print(" | GyZ = "); Serial.println(GyZ);  delay(333);  }   * MPU 6050 – Kalman 필터를 적용   #include <Wire.h>  #include "Kalman.h" // Source: https://github.com/TKJElectronics/KalmanFilter  #define RESTRICT\_PITCH // Comment out to restrict roll to ±90deg instead - please read: http://www.freescale.com/files/sensors/doc/app\_note/AN3461.pdf  Kalman kalmanX; // Kalman 인스턴스 생성  Kalman kalmanY;  /\* IMU 데이터 \*/  double accX, accY, accZ; // 가속도계 데이터  double gyroX, gyroY, gyroZ; // 자이로스코프 데이터  int16\_t tempRaw; // 온도 데이터  double gyroXangle, gyroYangle; // 자이로만을 사용하여 계산된 각도  double compAngleX, compAngleY; // 보완 필터를 사용하여 계산된 각도  double kalAngleX, kalAngleY; // 칼만 필터를 사용하여 계산된 각도  uint32\_t timer;  uint8\_t i2cData[14]; // I2C 데이터를 저장하는 버퍼  // TODO: Make calibration routine  void setup() {  Serial.begin(115200); // 시리얼 통신 시작  Wire.begin();// Wire 라이브러리 초기화  TWBR = ((F\_CPU / 400000L) - 16) / 2; // I2C 주파수를 400kHz로 설정  i2cData[0] = 7; // 샘플 레이트를 1000Hz로 설정 - 8kHz/(7+1) = 1000Hz  i2cData[1] = 0x00; // FSYNC 비활성화 및 260 Hz의 가속도 필터링, 256 Hz의 자이로스코프 필터링, 8 KHz 샘플링 설정  i2cData[2] = 0x00; // ±250deg/s의 자이로스코프 전체 범위로 설정  i2cData[3] = 0x00; // ±2g의 가속도계 전체 범위로 설정  while (i2cWrite(0x19, i2cData, 4, false)); // 4개 레지스터에 동시에 쓰기  while (i2cWrite(0x6B, 0x01, true)); // PLL 사용 및 X축 자이로스코프 참조 및 슬립 모드 비활성화  while (i2cRead(0x75, i2cData, 1)); // "WHO\_AM\_I" 레지스터 읽기  if (i2cData[0] != 0x68) { // 센서 읽기 오류 시  Serial.print(F("Error reading sensor"));  while (1);  }  delay(100); // 센서 안정화 대기  /\* 칼만 및 자이로 시작 각도 설정 \*/  while (i2cRead(0x3B, i2cData, 6)); // 센서에서 가속도계 데이터 읽기  accX = (i2cData[0] << 8) | i2cData[1];  accY = (i2cData[2] << 8) | i2cData[3];  accZ = (i2cData[4] << 8) | i2cData[5];  // Source: http://www.freescale.com/files/sensors/doc/app\_note/AN3461.pdf eq. 25 and eq. 26  // atan2 함수는 -π에서 π(radian)까지의 값을 출력합니다 - http://en.wikipedia.org/wiki/Atan2  // 그런 다음 radian 값을 degree로 변환합니다  #ifdef RESTRICT\_PITCH // Eq. 25 and 26  double roll = atan2(accY, accZ) \* RAD\_TO\_DEG;  double pitch = atan(-accX / sqrt(accY \* accY + accZ \* accZ)) \* RAD\_TO\_DEG;  #else // Eq. 28 and 29  double roll = atan(accY / sqrt(accX \* accX + accZ \* accZ)) \* RAD\_TO\_DEG;  double pitch = atan2(-accX, accZ) \* RAD\_TO\_DEG;  #endif  kalmanX.setAngle(roll); // 시작 각도 설정  kalmanY.setAngle(pitch);  gyroXangle = roll;  gyroYangle = pitch;  compAngleX = roll;  compAngleY = pitch;  timer = micros();  }  void loop() {  /\* 값 업데이트 \*/  while (i2cRead(0x3B, i2cData, 14)); // 센서에서 데이터 읽기  accX = ((i2cData[0] << 8) | i2cData[1]);  accY = ((i2cData[2] << 8) | i2cData[3]);  accZ = ((i2cData[4] << 8) | i2cData[5]);  tempRaw = (i2cData[6] << 8) | i2cData[7];  gyroX = (i2cData[8] << 8) | i2cData[9];  gyroY = (i2cData[10] << 8) | i2cData[11];  gyroZ = (i2cData[12] << 8) | i2cData[13];  double dt = (double)(micros() - timer) / 1000000; // Δ시간 계산  timer = micros();  // Source: http://www.freescale.com/files/sensors/doc/app\_note/AN3461.pdf eq. 25 and eq. 26  http://en.wikipedia.org/wiki/Atan2  // 가속도계를 사용하여 계산된 각도  #ifdef RESTRICT\_PITCH // Eq. 25 and 26  double roll = atan2(accY, accZ) \* RAD\_TO\_DEG;  double pitch = atan(-accX / sqrt(accY \* accY + accZ \* accZ)) \* RAD\_TO\_DEG;  #else // Eq. 28 and 29  // 가속도계를 사용하여 계산된 각도  double roll = atan(accY / sqrt(accX \* accX + accZ \* accZ)) \* RAD\_TO\_DEG;  double pitch = atan2(-accX, accZ) \* RAD\_TO\_DEG;  #endif  double gyroXrate = gyroX / 131.0; // deg/s로 변환  double gyroYrate = gyroY / 131.0; // deg/s로 변환  #ifdef RESTRICT\_PITCH  // 가속도계의 각도가 -180도에서 180도로 변할 때의 전환 문제를 해결  if ((roll < -90 && kalAngleX > 90) || (roll > 90 && kalAngleX < -90)) {  kalmanX.setAngle(roll);  compAngleX = roll;  kalAngleX = roll;  gyroXangle = roll;  } else  kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // 칼만 필터를 사용하여 각도 계산  if (abs(kalAngleX) > 90)  gyroYrate = -gyroYrate; // 가속도계 값이 제한되도록 속도를 반전시킵니다.  kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt);  #else  // 가속도계의 각도가 -180도에서 180도로 변할 때의 전환 문제를 해결  if ((pitch < -90 && kalAngleY > 90) || (pitch > 90 && kalAngleY < -90)) {  kalmanY.setAngle(pitch);  compAngleY = pitch;  kalAngleY = pitch;  gyroYangle = pitch;  } else  kalAngleY = kalmanY.getAngle(pitch, gyroYrate, dt); // 칼만 필터를 사용하여 각도 계산  if (abs(kalAngleY) > 90)  gyroXrate = -gyroXrate; // 가속도계 값이 제한되도록 속도를 반전시킵니다.  kalAngleX = kalmanX.getAngle(roll, gyroXrate, dt); // 칼만 필터를 사용하여 각도 계산  #endif  gyroXangle += gyroXrate \* dt; // 필터 없이 자이로 각도 계산  gyroYangle += gyroYrate \* dt;  //gyroXangle += kalmanX.getRate() \* dt; // 편향되지 않은 속도를 사용하여 자이로 각도 계산  //gyroYangle += kalmanY.getRate() \* dt;  compAngleX = 0.93 \* (compAngleX + gyroXrate \* dt) + 0.07 \* roll; // 보완 필터를 사용하여 각도 계산  compAngleY = 0.93 \* (compAngleY + gyroYrate \* dt) + 0.07 \* pitch;  // 자이로 각도가 너무 많이 드리프트 됐을 때 재설정  if (gyroXangle < -180 || gyroXangle > 180)  gyroXangle = kalAngleX;  if (gyroYangle < -180 || gyroYangle > 180)  gyroYangle = kalAngleY;  /\* 데이터 출력 \*/  #if 0 // 1로 설정하여 활성화  Serial.print(accX); Serial.print("\t");  Serial.print(accY); Serial.print("\t");  Serial.print(accZ); Serial.print("\t");  Serial.print(gyroX); Serial.print("\t");  Serial.print(gyroY); Serial.print("\t");  Serial.print(gyroZ); Serial.print("\t");  Serial.print("\t");  #endif  Serial.print(roll); Serial.print("\t");  Serial.print(gyroXangle); Serial.print("\t");  Serial.print(compAngleX); Serial.print("\t");  Serial.print(kalAngleX); Serial.print("\t");  Serial.print("\t");  Serial.print(pitch); Serial.print("\t");  Serial.print(gyroYangle); Serial.print("\t");  Serial.print(compAngleY); Serial.print("\t");  Serial.print(kalAngleY); Serial.print("\t");  #if 0 // 1로 설정하여 온도 출력  Serial.print("\t");  double temperature = (double)tempRaw / 340.0 + 36.53;  Serial.print(temperature); Serial.print("\t");  #endif  Serial.print("\r\n");  delay(2);  } |
| **활동 사진** | *※ 활동하고 있는 모습, 회로도, 완성된 모습, 실행화면 등..*  <MPU6050 회로도 참고> <IMU 회로도 참고>  실행결과 |
| **활동 결과** | *※ 활동 내용을 통해 나온 결과 정리*  MPU6050과 IMU를 사용하여 중력가속도 값을 계산하여 각도를 측정하고, 시리얼모니터로 측정값을 보여준다. 여기서 센서에서 얻은 가속도 값을 적분하여 시간에 따라 **오차가 누적된다. 따라서 오차와 노이즈의 문제를 줄이기 위해 상보필터, 칼만필터가 사용된다.** |
| **계획** | *※ 다음 활동 시 해야할 일*  MPU 6050에 Kalman 필터를 적용한 코드를 활용해 수정하여 실행시켜 보고, 상보필터에 관하여 더 찾아본 후 적용해 보겠습니다. |

2024년 4 월 12 일

작성자 : 장현주 (인)