M5StickC Plus2를 활용한 다기능 미니 **4WD** 경주 보조 장치 **“MPAS-FE”** 분석 보고서

1. 서론

M5StickC Plus2는 ESP32를 기반으로 하는 소형 개발 보드로, 디스플레이, 버튼, 마이크, 관성 측정 장치(IMU), 배터리와 같은 다양한 주변 장치를 통합하고 있어 임베디드 시스템 개발에 널리 사용됩니다. 본 보고서는 제공된 아두이노 C++ 코드를 분석하여 미니 4WD 경주 애호가를 위한 다기능 보조 장치의 구현을 목표로 하는 프로젝트를 상세히 살펴봅니다.

이 코드는 센서 데이터 수집, 실시간 분석 및 다양한 유틸리티 모드를 통합하여 모터 속도 추정, 경주 시간 측정, 섀시 정렬, 배터리 모니터링 및 관련 온라인 리소스에 대한 빠른 액세스를 제공합니다. 본 보고서는 코드의 구현 세부 사항, 잠재적인 개선 사항 및 코드에서 입증된 모범 사례에 관심 있는 기술적 배경을 가진 독자를 대상으로 합니다.

2. 코드 아키텍처 및 구성

- **2.1.** 헤더 파일
  제공된 코드에는 다음과 같은 여러 헤더 파일이 포함되어 있습니다.

  M5StickCPlus2.h는 M5StickC Plus2 하드웨어와의 상호 작용에 필요한 라이브러리를 제공하며, 여기에는 디스플레이, 버튼 및 기타 주변 장치에 대한 인터페이스가 포함됩니다.

  arduinoFFT.h는 고속 푸리에 변환(FFT) 연산을 가능하게 하여 주파수 분석 기능을 제공합니다. math.h는 코드에서 사용되는 표준 수학 함수를 포함합니다.

  battery_monitor.h는 고급 배터리 모니터링을 위한 사용자 정의 함수를 포함할 가능성이 높지만, 제공된 코드 스니펫에서는 기본적인 기능만 보여줍니다.

  buzzer.h는 M5StickC Plus2의 부저를 제어하기 위한 함수를 포함합니다. 이러한 라이브러리의 포함은 하드웨어 제어, 신호 처리 및 수학 연산을 위한 기존 기능을 활용하는 모듈식 접근 방식을 나타냅니다. 이는 코드 재사용성을 높이고 개발 시간을 단축시키는 데 기여합니다.

- **2.2.** 전역 변수 선언
  코드에는 여러 전역 변수가 선언되어 있으며, 각 변수는 특정 목적을 수행합니다.

  WHITE, BLUE 등과 같은 색상 정의는 전처리기 지시문을 사용하여 색상 코드를 설명적인 이름으로 나타냄으로써 코드의 가독성과 유지 관리성을 향상시킵니다.

  M5Canvas sprite 객체는 디스플레이에서 깜박임을 방지하고 시각적 경험을 개선하는 데 도움이 되는 더블 버퍼링을 위해 사용됩니다.

  버튼 처리 변수(btnAPressStart, longPressTriggered 등)는 모드 내에서 다양한 기능을 위해 짧게 누르기와 길게 누르기를 감지하는 데 사용됩니다.

  FFT 관련 변수(SAMPLES, SAMPLING_FREQUENCY, vReal, vImag, micBuffer, FFT)는 FFT 구성 매개변수(샘플 수 및 샘플링 주파수)와 FFT의 실수부와 허수부, 마이크 입력 및 FFT 객체를 저장하는 데 사용되는 버퍼를 자세히 설명합니다.

  FFT 분석을 위한 샘플 수는 *512*로 정의되어 있고, 샘플링 주파수는 *700Hz*로 설정되어 있습니다. 이는 주파수 분해능(높은 샘플 수 = 더 나은 분해능)과 처리 시간(더 많은 샘플 수 = 더 긴 처리 시간) 사이의 절충을 나타냅니다.

  상대적으로 낮은 샘플링 주파수는 분석을 미니 4WD의 모터 소음 분석에 적합한 낮은

오디오 주파수로 제한합니다. vReal 및 vImag 배열은 각각 FFT의 실수부와 허수부를 저장하는 데 사용되며, micBuffer는 원시 오디오 데이터를 저장하고, FFT 객체는 arduinoFFT 라이브러리의 인스턴스입니다.

샘플 수와 샘플링 주파수는 *FFT* 분석의 특성을 직접적으로 결정 합니다. 더 많은 샘플 수는 더 미세한 주파수 분해능을 제공하여 피크를 더 정확하게 식별할 수 있도록 합니다. 그러나 계산 부하도 증가합니다. 나이퀴스트-섀넌 표본화 정리에 따르면 샘플링 주파수는 관심 있는 최대 주파수의 최소 두 배여야 합니다. 여기서 700Hz의 샘플링 주파수는 최대 350Hz까지 분석할 수 있도록 하는데, 이는 미니 4WD 모터의 사운드를 분석하는 데 적합해 보입니다.

모드 제어 변수(mode, prevMode)는 장치의 다양한 작동 모드를 관리하고 모드 전환 중에 디스플레이 업데이트를 최적화하는 데 사용됩니다.

모드별 변수(예: maxFrequency, wheelSizes, stopwatchRunning, clockStartTime, compRoll, qrCodes)는 각 특정 모드와 연결되어 있으며, 저장하는 데이터와 해당 모드 내에서의 목적을 나타냅니다. 예를 들어, mode는 현재 활성화된 기능을 결정하고, wheelSizes는 센서 모드에서 속도 계산을 위한 다양한 휠 직경을 제공하며, stopwatchRunning은 스톱워치가 활성 상태인지 여부를 추적합니다.

각 작동 모드에 대해 고유한 변수 세트를 사용하는 것은 코드 구성이 잘 되어 있음을 보여주며, 이름 충돌을 방지하고 각 독립적인 기능의 상태를 더 쉽게 관리할 수 있도록 합니다. mode 변수는 중앙 상태 머신 컨트롤러 역할을 하여 loop() 함수 내에서 실행 흐름을 적절한 모드별 코드 블록으로 안내합니다. prevMode 변수는 모드 전환을 감지하고 화면 지우기와 같은 필요한 작업을 트리거하는 데 사용되는 영리한 최적화 방법으로, 효율성과 사용자 경험을 향상시킵니다.

- **2.3.** 함수 정의
  코드에는 setup(), loop(), getBatteryPercent(), applySMAFilter(), applyEMAFilter(), updateFilteredAngles(), kalmanFilter()와 같은 여러 함수가 정의되어 있습니다.

setup() 함수는 프로그램 시작 시 한 번 실행되며 하드웨어 주변 장치를 설정하고 초기 상태를 구성하는 데 사용됩니다.

loop() 함수는 setup()이 완료된 후 지속적으로 실행되며 프로그램의 주요 실행 주기를 형성합니다.

getBatteryPercent() 함수는 배터리 잔량을 계산하는 데 사용됩니다.

applySMAFilter(), applyEMAFilter() 및 **updateFilteredAngles()** 함수는 센서

데이터에 스무딩 필터를 적용하는 데 사용되며, **kalmanFilter()** 함수는 칼만 필터를 구현하여 롤 및 피치 각도를 추정합니다.

battery_monitor.h에서 **readStableVoltage()** 함수가 호출되어 안정적인 배터리 전압 판독값을 얻고, **buzzer.h**에서 **initBuzzer()** 함수가 호출되어 부저를 초기화합니다.

코드는 표준 아두이노 setup() 및 loop() 함수를 중심으로 구성되어 있으며, 배터리 모니터링, 필터링 및 칼만 필터링과 같은 특정 작업을 위한 추가 도우미 함수가 있습니다. 이러한 모듈식 설계는 코드 구성 및 가독성을 향상시킵니다. setup() 및 loop()을 중심으로 아두이노 프로그램의 기본 구조가 전개됩니다. 다른 함수의 존재는 전체 기능이 더 작고 관리하기 쉬운 단위로 논리적으로 분해되었음을 시사하며, 이는 잘 구조화된 코드의 특징입니다.

3. 작동 모드 상세 분석

- **3.1.** 모드 **0:** 센서 모드

기능**:** FFT를 이용한 실시간 주파수 측정, 사용자 선택 휠 크기 및 기어비를 기반으로 현재 및 최대 속도 추정, 배터리 잔량과 함께 이러한 값 표시. 최대 주파수 판독값 고정 기능 포함.

버튼 **A** 처리**:**

짧게 누르기(< 1000ms): 기어비 순환.

중간 누르기(1000ms <= 지속 시간 < 2000ms): 휠 크기 순환.

길게 누르기(>= 2500ms): 최대 주파수 고정 플래그 토글.

이 로직을 통해 사용자는 정확한 속도 추정을 위해 미니 4WD의 물리적 매개변수를 구성할수 있습니다.

고정 기능은 실행 중에 달성된 최고 주파수를 캡처하는 데 유용하며, 성능 측정에서 흔히 발생하는 요구 사항인 최고 달성 값을 캡처하는데 대한 해결책을 제시합니다. 성능 분석을 위한 귀중한 기능입니다.

버튼 A의 다중 누르기 기능은 이 모드 내에서 여러 구성 옵션에 액세스할 수 있는 간결한 방법을 제공합니다. 버튼 누르기의 다양한 지속 시간을 고유한 작업에 매핑하여 M5StickC Plus2의 제한된 버튼 수를 효율적으로 사용합니다.

마이크 입력 및 **FFT** 분석**:** 오디오 데이터를 micBuffer에 기록하고, FFT 처리를 위해 vReal에 복사하고, 해밍 윈도우를 적용하고, FFT를 계산하고, 복소수 결과를 크기로 변환합니다. 이것이 센서 모드의 핵심으로, 장치가 모터에서 발생하는 소리를 분석하여 회전 주파수를 결정할 수 있도록 합니다. 스펙트럼 누출을 줄이기 위해 해밍 윈도우가 적용됩니다.

FFT를 사용하면 미니 4WD와 같은 움직이는 차량을 분석하는 데 상당한 이점인 모터의 회전 주파수를 비접촉 방식으로 측정할 수 있습니다. 해밍 윈도우를 선택한 것은 주파수 추정의 정확도를 향상시키기 위한 일반적인 신호 처리 기술에 대한 이해를 보여줍니다. 오디오 녹음, 윈도우 함수 적용, FFT 수행, 크기 계산 과정은 주파수 영역 분석의 표준

절차입니다. 해밍 윈도우는 사이드 로브 레벨을 효과적으로 줄여 더 깨끗한 주파수 스펙트럼을 생성하므로 스펙트럼 분석에 일반적으로 사용됩니다.

주파수 및 속도 계산**:** FFT.majorPeak()는 주요 주파수를 찾는 데 사용됩니다. 그런 다음 이 주파수는 선택한 휠 크기 및 기어비를 사용하여 RPM으로 변환되고 최종적으로 속도로 변환됩니다.

이 섹션에서는 측정된 주파수를 사용자에게 의미 있는 메트릭인 분당 회전수(RPM) 및 시간당 킬로미터(km/h)로 변환합니다. 공식은 휠 직경과 기어비를 정확하게 고려합니다. 주파수를 속도로 변환하면 데이터가 미니 4WD 성능에 대한 사용자의 관심과 더 직접적으로 관련됩니다.

round() 사용과 int로의 캐스팅은 이해하기 쉬운 정수 값으로 RPM을 표시하는 데 중점을 둔다는 것을 시사합니다. 매직 넘버 0.06은 단위 변환(mm에서 km, 분에서 시간)을 결합한 것 같습니다.

속도 계산에 사용된 공식은 기본적인 물리 법칙을 기반으로 합니다. 속도는 휠의 둘레와 회전 속도에 비례 합니다.

기어비는 정의 방식에 따라 승수 또는 제수 역할을 합니다. (여기서는 휠에서의 RPM을 줄이는 것으로 보입니다) 상수 0.06은 입력 단위(Hz 단위의 주파수, 미터 단위의 직경, 무단위의 기어비)에서 km/h 단위의 속도를 얻는 데 필요한 변환 계수를 포함할 가능성이 높습니다.

디스플레이 업데이트**:** 스프라이트 객체를 사용하여 휠 크기, 배터리 잔량, 최대 및 현재 피크 주파수, 현재 및 최대 추정 속도 및 RPM 값을 화면에 그립니다.

sprite.fillRect() 및 sprite.drawString() 사용은 계산된 센서 데이터로 디스플레이를 업데이트하기 위해 더블 버퍼링 기술이 어떻게 사용되는지 보여줍니다.

배터리 부족 시 조건부로 깜박이는 배터리 표시기는 명확한 시각적 경고를 제공합니다. 화면의 관련 부분을 지우고 다시 그리는 것은 디스플레이가 아티팩트 없이 깨끗하게 업데이트되도록 보장합니다. 텍스트 요소의 전략적 배치와 서식은 이 모드의 사용자 인터페이스에 기여합니다. 깜박이는 배터리 표시기는 중요한 저전력 상태를 사용자에게 알리는 표준적이고 효과적인 방법입니다.

주요 내용**:** 실시간 주파수 분석을 사용하여 모터 속도를 추정합니다. 사용자는 휠 크기와 기어비를 구성할 수 있습니다. 최대 주파수와 속도를 고정할 수 있습니다.

숨겨진 연결**:** 속도 추정의 정확도는 사용자가 휠 크기와 기어비를 올바르게 입력하는지에 크게 좌우됩니다. 이러한 입력에 오류가 있으면 계산된 속도에 직접적인 오류가 발생합니다. 모터의 주요 회전 속도에 직접적으로 해당하는 주요 주파수라는 가정이 항상 사실이 아닐 수 있습니다**.** 특히 모터 소리에 상당한 고조파 또는 기타 노이즈 성분이 있는 경우가 그러합니다.

광범위한 의미**:** 이 모드는 미니 4WD 애호가에게 다양한 설정에서 모터 성능을 정량적으로 평가할 수 있는 귀중한 도구를 제공합니다. 고정 기능은 경주 중에 달성된 최고 성능을 확인하는 데 사용할 수 있습니다.

표 **1:** 속도 계산 단위 및 변환 계수

수량	단위	변환 계수
휠 직경	mm	/ 1000 (미터로)
RPM	회전/분	/ 60 (Hz로)
둘레	미터	π * 직경
속도	km/h	* 3.6 (m/s에서 km/h로)
결합된 변환 계수		0.06

- **3.2.** 모드 **1:** 진행률 표시줄 모드

  기능**:** 모터의 주파수를 측정하고 다양한 모터 유형에 대해 미리 정의된 최대 임계값과 비교하여 수평 진행률 표시줄로 표시합니다.

  모터 구성**:** Motor 구조체 배열은 다양한 모터 유형에 대한 이름, 최대 임계 주파수 및 색상을 정의합니다. 이 데이터 구조를 통해 코드는 특정 성능 특성(최대 주파수)을 다양한 모터 모델과 연결하여 비교 시각화를 할 수 있습니다. 이 모드는 미니 4WD 모터에 특화되어 있으며, 모터의 작동 주파수가 예상 최대 주파수에 비해 어느 정도인지 빠르게 시각적으로 보여줍니다. 이는 모터가 정상 범위 내에서 작동하는지 여부를 식별하는 데 유용할 수 있습니다. Motor 구조체는 이 모드에 대해 다양한 모터 유형을 나타내는 데 필요한 주요 정보를 효과적으로 캡슐화합니다. 배열을 사용하면 향후 더 많은 모터 모델을 포함하도록 쉽게 확장할 수 있습니다.

  버튼 **A** 처리**:** 짧게 누르면 사용 가능한 모터 유형이 순환됩니다. 이 간단한 버튼 누르기를 통해 사용자는 분석하려는 모터를 선택할 수 있습니다. 모터 선택의 순환적인 특성은 사용자가 미리 정의된 모든 모터 프로필에 쉽게 액세스할 수 있도록 보장합니다. 모듈로 연산자(%)를 사용하는 것은 배열을 통해 원형 탐색을 만드는 표준적인 방법으로, 끝에 도달한 후 인덱스가 시작 부분으로 다시 돌아가도록 합니다.

  주파수 측정 및 진행률 표시줄**:** 모드 0과 유사한 FFT 분석입니다. 측정된 주파수는 선택한 모터의 maxThreshold에 대해 정규화되어 진행률 표시줄의 채우기 백분율을 결정합니다. 이 시각화는 현재 모터 주파수가 예상 최대 주파수와 어떻게 비교되는지 직관적으로 보여줍니다. 진행률 표시줄은 모드 0에 표시된 숫자 값에 비해 모터 성능에 대한 더 정성적인 평가를 제공합니다.

  측정된 주파수가 임계값보다 높으면 진행률 표시줄이 화면 너비를 초과하는 것을 방지하기 위해 비율을 1.0으로 제한합니다. 비율에 적용된 스무딩은 진행률 표시줄의

시각적 지터를 줄일 가능성이 높습니다. 주파수를 0에서 1 사이의 범위로 정규화하면 진행률 표시줄의 너비에 쉽게 매핑할 수 있습니다.

모터의 색상을 진행률 표시줄에 사용하면 선택한 모터 유형과 시각적으로 연결됩니다.

디스플레이 업데이트**:** 화면을 지우고 선택한 모터에 해당하는 색상으로 진행률 표시줄을 그립니다. 진행률 표시줄 아래에 모터 이름, 최대 임계값 및 현재 주파수를 표시합니다. 이는 시각적 진행률 표시줄과 함께 명확한 텍스트 정보를 제공합니다.

시각적 표현(진행률 표시줄)과 텍스트 정보(모터 이름, 주파수)를 결합하면 사용자의 데이터 이해도가 높아집니다.

sprintf를 사용하는 것은 C/C++에서 표시 목적으로 문자열을 포맷하는 일반적인 방법으로, 모터 이름과 주파수 값을 동적으로 삽입할 수 있습니다.

주요 내용**:** 최대값에 대한 모터 주파수의 시각적 표현. 다양한 모터 유형에 대한 사전 정의된 프로필.

숨겨진 연결**:** 이 모드의 효과는 motors 배열에 정의된 각 모터의 maxThreshold 값의 정확성에 따라 달라집니다. 잘못된 임계값은 오해를 불러일으키는 진행률 표시줄 표현으로 이어질 수 있습니다. 비율에 적용된 스무딩 계수는 모터 주파수 변화에 대한 진행률 표시줄의 응답성에 영향을 미칠 수 있습니다.

광범위한 의미**:** 이 모드는 모터가 예상대로 작동하는지 여부 또는 문제가 있는지 여부(예: 예상 최대 주파수보다 훨씬 낮게 작동)를 빠르게 평가하는 데 유용할 수 있습니다. 또한 다양한 모터의 성능을 예상 벤치마크와 비교하는 데 사용할 수도 있습니다.

표 **2:** 모터 프로필

모터 이름	최대 임계값 **(Hz)**	색상
F130	400	WHITE
Rev	430	BLUE
Torque	420	ORANGE

Atomic	420	DARKGRAY
Light	470	YELLOW
Hyper	550	RED
Power	630	DARKGREEN
Mach	630	DARKRED
Sprint	650	LIGHTGRAY
Ultra	700	PURPLE

- **3.3.** 모드 **2:** 스톱워치 모드

  기능**:** 랩 타이밍 기능이 있는 디지털 스톱워치입니다. 랩 목표(2, 3 또는 5랩)를 설정할 수 있으며 목표에 도달하면 시각적 표시를 제공합니다.

  버튼 **A** 처리**:**

  짧게 누르기: 시작 또는 랩 기록. 스톱워치가 정지되고 랩 목표에 도달한 경우 스톱워치를 다시 시작합니다.

  길게 누르기(>= 1000ms, 스톱워치 정지됨): 랩 목표(2, 3, 5)를 순환합니다.

이를 통해 사용자는 스톱워치 기능을 쉽게 제어하고 타이밍 요구 사항에 맞게 랩 목표를 사용자 지정할 수 있습니다.

버튼 A를 짧게 누르기와 길게 누르기로 구분하여 기본 스톱워치 기능(시작/랩)과 보조 구성 옵션(랩 목표)에 모두 액세스할 수 있습니다. 목표에 도달한 후 자동으로 재설정되면 반복적인 타이밍 세션이 간소화됩니다.

플래그(longPressTriggered)를 사용하면 길게 누르기가 감지된 직후에 짧게 누르기 동작이 트리거되는 것을 방지합니다. 모듈로 연산자는 랩 목표가 허용된 값을 순환하도록 보장합니다.

버튼 **B** 처리**:**

짧게 누르기: 다음 모드로 순환합니다.

중간 누르기(1000ms <= 지속 시간 < 2000ms): 이전 모드로 순환합니다.

길게 누르기(>= 3000ms): 스톱워치를 재설정합니다(정지, 시간 지우기).

표준 스톱워치 제어(재설정) 및 모드 간 탐색을 제공합니다.

버튼 B의 다양한 누르기 지속 시간은 스톱워치 및 전체 장치 작동을 위한 포괄적인 제어 옵션을 제공합니다. 버튼 누르기의 다양한 시간 임계값을 사용하면 단일 버튼에 여러 개의 고유한 동작을 매핑하여 더 많은 물리적 버튼 없이 기능을 향상시킬 수 있습니다.

시간 기록 및 랩 기록**:** millis()를 사용하여 경과 시간을 추적하고 랩 시간을 lapTimes 배열에 저장합니다. 기록되는 랩 수에 제한(MAX_LAPS)을 구현합니다. 이것이 스톱워치 기능의 핵심 로직으로, 랩 시간을 정확하게 측정하고 저장합니다.

millis()를 사용하면 아두이노에서 시간 간격을 비교적 정확하게 측정할 수 있습니다.

MAX_LAPS 제한은 잠재적인 버퍼 오버플로를 방지합니다. 스톱워치가 시작되면 stopwatchStartTime은 현재 시간을 밀리초 단위로 기록합니다. 랩 기록 중에는 마지막 랩(또는 시작) 이후 경과된 시간이 lastLapTime에서 현재 시간을 빼서 계산됩니다. 이 lapInterval은 lapTimes 배열에 저장되고 lastLapTime이 업데이트됩니다. lapCount 변수는 기록된 랩 수를 추적하고 if (lapCount < MAX_LAPS) 조건은 배열이 오버플로되지 않도록 보장합니다.

랩 목표 및 시각적 피드백**:** 기록된 랩 수가 splitTarget에 도달하면 타이머 표시가 깜박이기 시작합니다. 이는 사용자가 목표 랩 수에 도달했음을 명확하게 시각적으로 나타냅니다.

깜박이는 효과는 특정 이벤트 또는 상태에 사용자의 주의를 끌기 위한 일반적이고 효과적인 방법입니다. 깜박임은 타이머(lastBlinkTime)와 정의된 간격(blinkInterval)으로 토글되는 상태 변수(blinkState)를 사용하여 구현되어 켜짐-꺼짐 효과를 생성합니다.

디스플레이 업데이트**:** 상단에 큰 글꼴로 기본 타이머를 표시하고 아래에 기록된 랩 시간을 나열합니다. 기본 타이머의 색상은 선택한 랩 목표에 따라 달라집니다.

스톱워치 데이터를 명확하고 체계적으로 표시합니다.

다양한 랩 목표에 대해 다양한 색상을 사용하면 현재 설정에 대한 즉각적인 시각적 단서를 사용자에게 제공합니다.

개별 랩 시간을 표시하면 상세한 성능 분석이 가능합니다. sprintf 함수는 시간 값(분, 초, 센트초)을 읽을 수 있는 문자열로 포맷하는 데 사용됩니다. 조건문을 사용하여 splitTarget 값에 따라 타이머 색상을 설정합니다.

주요 내용**:** 랩 타이밍 기능이 있는 기능성 스톱워치. 목표 도달 시 시각적 피드백이 있는 구성 가능한 랩 목표.

숨겨진 연결**:** 기록할 수 있는 최대 랩 수는 lapTimes 배열의 크기(MAX_LAPS)에 의해 제한됩니다. 사용자가 더 많은 랩을 기록하려고 하면 이전 랩 시간이 덮어쓰여지거나 기록이 중지될 수 있습니다. 랩 타이밍의 정확도는 millis() 함수의 해상도와 버튼 누르기의 응답성에 따라 달라집니다.

광범위한 의미**:** 이 모드는 미니 4WD 경주 시간을 측정하는 데 직접적으로 유용하며, 경주자가 자신의 성능을 추적하고 랩 시간을 분석할 수 있도록 합니다. 랩 목표 기능은 훈련 또는 특정 경주 형식에 사용할 수 있습니다.

- **3.4.** 모드 **3:** 경과 시간 시계 모드

  기능**:** 모드가 활성화된 이후 또는 재설정 이후의 지속 시간을 표시하는 경과 시간 시계를 표시합니다. 일시 정지/재개 기능 포함.

  버튼 **A** 처리**:**

  짧게 누르기: 시계 일시 정지와 재개 사이를 전환합니다.

  길게 누르기(>= 1000ms): 경과 시간을 0으로 재설정합니다.

경과 시간 시계에 대한 기본 제어를 제공합니다. 재설정을 위한 길게 누르기와 일시 정지/재개를 위한 짧게 누르기는 시계 작동에 대한 직관적인 제어를 제공합니다.

스톱워치 모드와 유사하게, 플래그(longPressTriggered)는 짧게 누르기와 길게 누르기를 구별하는 데 사용됩니다. 길게 누르기가 감지되면 clockStartTime이 현재 시간으로 재설정되고 pausedTime 및 clockPaused 변수도 재설정됩니다. 짧게 누르면 clockPaused의 상태가 토글됩니다.

시계가 일시 정지되면 그때까지의 경과 시간이 pausedTime에 저장됩니다. 재개되면 clockStartTime이 일시 정지된 시간을 고려하여 조정됩니다.

시간 계산**:** clockStartTime 및 현재 시간을 기준으로 시간, 분, 초 단위로 경과 시간을 계산합니다. pausedTime을 사용하여 일시 정지 상태를 처리합니다. 이 로직은 일시 정지 상태를 고려하여 경과 시간을 정확하게 계산합니다. 정수 나눗셈과 모듈로 연산자를 사용하면 총 경과 시간(밀리초)에서 시간, 분, 초를 효과적으로 추출할 수 있습니다. 총 경과 시간은 시계가 일시 정지되었는지 여부에 따라 다르게 계산됩니다. clockPaused가 참이면 저장된 pausedTime 값(일시 정지 전까지 누적된 경과 시간)이 사용됩니다. 그렇지 않으면 현재 millis()에서 clockStartTime을 빼서 현재 경과 시간을 계산합니다. 그런 다음 총 경과 시간(밀리초)은 적절한 변환 계수(1시간에 3600000밀리초, 1분에 60000밀리초, 1초에 1000밀리초)를 사용하여 정수 나눗셈과 모듈로 연산자를 통해 시간, 분, 초로 변환됩니다.

디스플레이 업데이트**:** 초 값이 변경될 때만 시간 표시를 업데이트하여 불필요한 화면 다시 그리기를 최소화합니다. 경과 시간을 "HH:MM:SS" 형식으로 화면 중앙에 표시합니다. 디스플레이 업데이트를 최적화하고 경과 시간을 명확하게 표시합니다. 필요한 경우에만 디스플레이를 업데이트하면 전력 소비가 줄어들고 불필요한 작업을 피하여 성능이 향상됩니다. 시간 표시를 중앙에 배치하면 쉽게 읽을 수 있습니다.

현재 초 값을 lastDisplaySecond와 비교하여 코드는 loop() 함수가 훨씬 빠르게 실행되더라도 1초에 한 번만 디스플레이가 업데이트되도록 합니다. MC_DATUM(중앙-중간 기준)을 사용하면 계산된 정확한 중앙 좌표에 텍스트가 그려집니다.

주요 내용**:** 일시 정지/재개 및 재설정 기능이 있는 간단한 경과 시간 카운터입니다. 최적화된 디스플레이 업데이트.

숨겨진 연결**:** 정확하게 추적할 수 있는 최대 경과 시간은 시간을 밀리초 단위로 저장하는 데 사용되는 unsigned long 변수의 롤오버(약 49.7일)에 의해 제한됩니다. 더 긴 시간 동안은 시계가 재설정됩니다. 정확도는 millis() 함수의 해상도에 따라 달라집니다.

광범위한 의미**:** 이 모드는 연습 주행 또는 피트 스톱 시간과 같은 미니 4WD 경주와 관련된 다양한 활동 시간을 측정하는 데 유용할 수 있습니다. 다른 작업에도 범용 타이머로 사용할 수 있습니다.

- **3.5.** 모드 **4:** 수평계 모드 **(**자이로 기반**)**

  기능**:** M5StickC Plus2의 내장 IMU를 사용하여 수평계 역할을 하며, 롤, 피치 및 요 각도를 표시합니다. 보상 필터, 저역 통과 필터(LPF), 지수 이동 평균(EMA) 및 칼만

필터를 사용하여 각도를 추정합니다. 기준선 보정을 통해 영점을 재설정할 수 있습니다.

버튼 **A** 처리**:**

길게 누르기(>= 1000ms): 현재 방향으로 기준 롤, 피치 및 요 각도를 재보정합니다.

이를 통해 사용자는 장치의 현재 방향을 기준으로 수평계의 영점을 쉽게 재설정할 수 있습니다.

이 보정 기능은 장치의 초기 기울기 또는 오프셋을 보상할 수 있으므로 실제 사용에 매우 중요합니다. 버튼을 충분히 오래 누르면 현재 계산된 롤, 피치 및 요 각도가 새로운 기준선 값으로 저장됩니다. 그 후의 각도 측정은 이러한 새로운 기준선에 상대적으로 이루어집니다.

IMU 데이터 획득**:** M5StickC Plus2의 IMU에서 가속도계 및 자이로스코프 데이터를 읽습니다. 이것이 각도 추정에 필요한 원시 센서 데이터를 얻는 기본적인 단계입니다. 코드는 가속도계와 자이로스코프 데이터를 모두 활용하며, 이는 더 정확하고 안정적인 방향 추정을 위해 센서 융합 기술에서 일반적입니다.

M5StickC Plus2 라이브러리에서 제공하는 M5.Imu 객체는 온보드 관성 측정 장치에서 직접 판독값을 액세스하는 메서드를 제공합니다. getAccelData()는 x, y 및 z축을 따라 가속도를 검색하고, getGyroData()는 이러한 축을 중심으로 각속도를 검색합니다. 이러한 원시 센서 값은 각도 추정 알고리즘의 입력으로 사용됩니다.

시간 델타 계산**:** 연속적인 센서 판독값 사이의 시간 차이(dt)를 계산합니다. 이는 자이로스코프 데이터를 통합하고 칼만 필터를 사용하는 데 필요합니다. 정확한 시간 측정은 적절한 센서 융합 및 필터링에 필수적입니다. millis()를 사용하여 시간 차이를 계산하면 샘플링 간격을 비교적 정확하게 측정할 수 있습니다. 초 단위로 변환(/ 1000.0)하면 후속 계산에 일관된 단위를 사용할 수 있습니다.

millis() 함수는 아두이노 보드가 현재 프로그램을 실행한 이후 경과된 밀리초 수를 반환합니다. 이전 IMU 판독값의 타임스탬프를 previousTimeCF에 저장하고 현재 타임스탬프(currentTimeCF)와 비교하여 두 판독값 사이의 시간 차이를 계산할 수 있습니다. 이 시간 차이 dt는 자이로스코프의 각속도를 통합하여 각 변위를 얻는 데 매우 중요하며 칼만 필터 방정식의 핵심 매개변수이기도 합니다. 1000.0으로 나누면 시간 차이가 밀리초에서 초로 변환되어 후속 계산에서 일관된 단위를 사용할 수 있습니다.

칼만 필터 구현**:** 가속도계 및 자이로스코프 데이터를 융합하여 롤 및 피치 각도를

추정하는 칼만 필터를 구현합니다. 칼만 필터는 시스템의 동적 모델과 노이즈가 많은 측정을 결합하여 시스템의 상태(이 경우 롤 및 피치 각도)를 최적으로 추정하는 정교한 알고리즘입니다. 칼만 필터를 사용하면 가속도계와 자이로스코프 데이터의 고유한 노이즈와 드리프트를 해결하여 각도 측정을 더 정확하고 안정적으로 만들 수 있습니다.

요 각도는 자이로스코프 데이터를 통합하여 추정합니다. 요 각도를 별도로 처리(통합만 적용)하는 것은 절대 참조가 없으면 드리프트가 발생하기 쉽기 때문에 일반적입니다. 필터 매개변수*(Q_angle, Q_bias, R_measure)*는 필터의 성능에 매우 중요하며 센서 특성에 따라 신중하게 조정해야 합니다.

kalmanFilter() 함수는 칼만 필터 알고리즘의 예측 및 업데이트 단계를 구현할 가능성이 높습니다. 새로운 각도 측정값(가속도계에서), 새로운 각속도(자이로스코프에서) 및 시간 차이(dt)를 입력으로 받습니다. 필터는 이러한 입력과 내부 상태(추정 각도 및 바이어스) 및 오류 공분산 행렬을 사용하여 업데이트된 각도 추정값을 생성합니다.

compRoll 및 compPitch 변수는 적절한 가속도계에서 파생된 각도와 자이로스코프 속도로 이 함수를 호출하여 업데이트됩니다.

compYaw 각도는 자이로스코프의 요 속도(gz)와 시간 차이(dt)의 곱을 단순히 더하여 업데이트되는데, 이는 단순 적분에 해당합니다.

디스플레이 업데이트**:** 화면을 지우고 보정된 롤, 피치 및 요 각도를 도 단위로 표시해 수평계 판독값을 사용자에게 제공합니다. 각도를 소수점 한 자리까지 표시하면 정밀도와 가독성 사이의 균형을 제공합니다.

롤, 피치 및 요에 대해 서로 다른 색상을 사용하면 구별하는 데 도움이 됩니다. 계산된 각도를 표시하기 전에 보정 중에 얻은 기준선 오프셋이 현재 compRoll, compPitch 및 compYaw 값에서 빼집니다. 이렇게 하면 표시된 각도가 보정된 영점 방향을 기준으로 합니다.

String() 함수는 부동 소수점 각도 값을 문자열로 변환하는 데 사용되며, 스프라이트 객체의 drawString() 메서드는 이러한 문자열을 레이블("R:", "P:", "Y:") 및 도 기호("\xB0")와 함께 화면에 표시하는 데 사용됩니다.

주요 내용**:** IMU를 사용하여 수평 감지. 롤 및 피치 추정을 위한 칼만 필터 구현. 기준선 보정 허용.

숨겨진 연결**:** 칼만 필터의 성능은 선택한 매개변수(Q_angle, Q_bias, R_measure)에 크게 좌우됩니다. 매개변수를 잘못 조정하면 각도 추정이 노이즈가 많거나 느려질 수 있습니다. 자이로스코프 통합에만 의존하는 요 각도 추정은 시간이 지남에 따라 드리프트가 발생하기 쉽습니다. 가속도계에서 파생된 각도의 정확도는 장치에 작용하는 유일한 가속도가 중력이라는 가정에 따라 달라집니다. 외부 가속도는 롤 및 피치 추정에

오류를 발생시킵니다.

광범위한 의미**:** 이 모드는 미니 4WD 섀시가 수평인지 확인하는 데 유용하며, 이는 최적의 성능을 위해 중요할 수 있습니다. 보정 기능을 통해 다양한 사용 시나리오에 적응할 수 있습니다. 칼만 필터를 사용하는 것은 더 간단한 방법보다 더 발전된 센서 데이터 융합 접근 방식을 보여줍니다.

표 **3:** 칼만 필터 매개변수

매개변수	값	설명
Q_angle	0.0005	각도에 대한 프로세스 노이즈 공분산 (각도가 얼마나 변할 수 있는지에 영향)
Q_bias	0.001	자이로 바이어스에 대한 프로세스 노이즈 공분산 (바이어스가 얼마나 빨리 보정되는지에 영향)
R_measure	0.05	측정 노이즈 공분산 (가속도계 판독값을 얼마나 신뢰하는지에 영향)

- **3.6.** 모드 **5:** 전압계 모드

  기능**:** 배터리 전압을 읽어 화면에 표시합니다.

  배터리 전압 판독**:** battery_monitor.h에 정의된 readStableVoltage() 함수를 호출하여 안정적인 배터리 전압 판독값을 얻습니다. 이는 안정적인 판독값을 제공하기 위해 일종의 평균화 또는 필터링을 통합하여 배터리 전압 측정을 처리하기 위한 별도의 모듈을 사용함을 나타냅니다.

  배터리 모니터링 로직을 전용 파일(battery_monitor.h)로 분리하면 코드 모듈성과 재사용성이 향상됩니다.

  함수 이름 readStableVoltage()는 노이즈 또는 일시적인 변동을 필터링하여 신뢰할

수 있는 전압 판독값을 제공하려는 노력을 시사합니다. 모드 5의 loop()에 있는 메인 코드는 단순히 readStableVoltage() 함수를 호출합니다. 이는 배터리 전압을 실제로 하드웨어에서 읽는 방법(아날로그-디지털 변환기 사용)과 판독값을 안정화하는 방법(예: 여러 샘플 평균화)에 대한 구현 세부 정보가 battery_monitor.h 및 해당 .cpp 파일 내에 캡슐화되어 있음을 의미합니다. 이러한 관심사 분리는 메인 코드를 더 깔끔하고 이해하기 쉽게 만듭니다.

디스플레이 업데이트**:** 화면을 지우고 배터리 전압을 소수점 세 자리까지 표시한 다음 단위 "V"를 표시합니다. 명확하고 정확한 배터리 전압 판독값을 제공합니다. 전압을 소수점 세 자리까지 표시하면 비교적 높은 수준의 정밀도를 제공하여 배터리 상태를 모니터링하거나 미묘한 전압 강하를 감지하는 데 유용할 수 있습니다.

sprite.fillScreen(BLACK) 명령은 디스플레이 버퍼를 지웁니다. 그런 다음 sprite.setTextSize(6)은 전압 판독값을 위한 큰 글꼴 크기를 설정하고 sprite.setCursor(20,45)는 텍스트 위치를 지정합니다.

sprite.print(batteryVoltage, 3) 명령은 부동 소수점 batteryVoltage 값을 소수점 이하 세 자리까지 표시하도록 포맷하여 표시합니다. 마지막으로 sprite.setTextSize(3) 및 sprite.print("V")는 전압 판독값 옆에 더 작은 글꼴로 단위 "V"를 표시합니다.

주요 내용**:** M5StickC Plus2의 배터리 전압을 표시합니다. 안정적인 판독을 위해 외부 함수를 사용합니다.

숨겨진 연결**:** 전압 판독값의 정확도는 M5StickC Plus2의 전압 감지 회로의 보정 및 설계와 readStableVoltage() 함수 내의 구현에 따라 달라집니다. 온도와 같은 요인도 배터리 전압 판독값에 영향을 미칠 수 있습니다.

광범위한 의미**:** 이 모드는 장치의 전원 상태를 모니터링하는 데 필수적이며, 사용자가 배터리를 충전해야 할 시기를 알 수 있도록 합니다. 배터리 관련 문제를 진단하는 데에도 도움이 될 수 있습니다.

- **3.7.** 모드 **6: QR** 코드 모드

기능**:** 미리 정의된 일련의 QR 코드를 화면에 표시하고, 사용자는 버튼 A를 사용하여 코드를 순환할 수 있습니다.

QR 코드 데이터**:** 문자 포인터 배열(qrCodes)은 QR 코드로 인코딩할 문자열을 저장하고, 다른 배열(qrLabels)은 각 QR 코드에 해당하는 레이블을 제공합니다. 이는 표시될 QR 코드의 내용과 설명을 정의합니다.

코드와 레이블에 대해 별도의 배열을 사용하면 각 QR 코드에 설명 텍스트를 쉽게 연결할 수 있습니다.

소스코드에 제공된 URL은 미니 4WD 애호가를 위한 관련 온라인 리소스에 대한 링크를 제안합니다.

qrCodes 배열은 QR 코드로 인코딩될 실제 데이터(이 경우 URL)를 보유합니다. qrLabels 배열은 각 해당 QR 코드에 대한 사람이 읽을 수 있는 설명을 제공하여 사용자가 스캔하기 전에 QR 코드에 어떤 정보가 포함되어 있는지 더 쉽게 이해할 수 있도록 합니다. const char*를 사용하는 것은 이것들이 메모리에 저장된 상수 문자열임을 나타냅니다.

버튼 **A** 처리**:** 짧게 누르면 시퀀스의 다음 QR 코드로 순환하여 사용 가능한 QR 코드를 탐색하는 간단한 방법을 제공합니다. 모듈로 연산자는 마지막 코드가 표시된 후 QR 코드 인덱스가 시작 부분으로 다시 돌아가도록 보장하여 연속 루프를 만듭니다. 버튼 A가 해제되면 currentQRIndex가 증가합니다. numQRs(총 QR 코드 수)로 모듈로 연산자(%)를 사용하면 인덱스가 항상 유효한 범위(0에서 numQRs - 1) 내에 유지됩니다. qrDisplayed 플래그는 false로 설정되어 loop() 함수의 다음 반복에서 새 QR 코드를 화면에 그려야 함을 나타냅니다.

디스플레이 업데이트**:** 화면을 지우고 현재 QR 코드를 중앙에 그리고 아래에 해당 레이블을 표시하여, 사용자에게 QR 코드와 설명을 제공합니다. QR 코드를 중앙에 배치하면 스캔하기가 더 쉬워집니다. 레이블은 QR 코드의 내용에 대한 컨텍스트를 제공합니다.

M5Stack 라이브러리의 sprite.qrcode() 함수는 QR 코드 데이터(qrCodes), QR 코드의 왼쪽 상단 모서리에 대한 x 및 y 좌표, QR 코드의 크기를 입력으로 사용합니다. 그런 다음 QR 코드를 스프라이트 버퍼에 그립니다. 현재 QR 코드에 해당하는 레이블은 qrLabels 배열에서 검색되어 sprite.drawString() 함수를 사용하여 QR 코드 아래에 그려집니다. 레이블의 좌표는 QR 코드 아래에 수평으로 중앙에 배치되도록 계산됩니다.

주요 내용**:** 미리 정의된 QR 코드를 표시합니다. 버튼 누르기로 코드를 순환합니다. 각 QR 코드에 대한 레이블을 포함합니다.
숨겨진 연결**:** 이 모드의 기능은 사용자의 스마트폰 또는 기타 장치에 QR 코드 스캐닝 앱이 있는지 여부에 따라 달라집니다. 이 모드의 유용성은 연결된 온라인 리소스가 사용자에게 얼마나 관련성이 있는지에 따라 결정됩니다. QR 코드의 크기와 인코딩된

데이터의 복잡성은 M5StickC Plus2 화면에서 QR 코드를 스캔할 수 있는지 여부에 영향을 미칠 수 있습니다.

광범위한 의미**:** 이 모드는 온라인 커뮤니티 링크, 경주 장소 지도 또는 제품 정보와 같은 정보를 다른 미니 4WD 애호가와 편리하게 공유할 수 있는 방법을 제공합니다. 최신 스마트폰의 QR 코드 스캐너의 보편성을 활용합니다.

- **3.8.** 모드 **7:** 버전 정보 모드

기능**:** 컴파일 날짜와 시간을 버전 문자열로 표시합니다.

버전 문자열 생성**:** __DATE__ 및 __TIME__ 전처리기 매크로에서 연도, 월, 일 및 시간을 추출하여 YYMMDDhhmmss 형식의 버전 문자열로 결합합니다. 이를 통해 사용자는 장치에서 실행 중인 펌웨어의 빌드 버전을 쉽게 식별할 수 있습니다.

컴파일 타임스탬프를 버전 문자열의 일부로 사용하면 각 빌드에 대한 고유한 식별자를 제공하므로 디버깅 및 변경 사항 추적에 유용합니다.

특정 형식(YYMMDDhhmmss)을 사용하면 버전의 시간순 정렬이 가능합니다. 월 약어를 숫자로 수동으로 매핑하는 것은 다른 컴파일러 또는 로케일에서 __DATE__ 매크로의 잠재적인 서식 불일치에 대한 해결 방법을 보여줍니다.

__DATE__ 및 __TIME__ 매크로는 컴파일 중에 컴파일러에 의해 자동으로 정의됩니다. 코드는 String 클래스의 substring() 메서드를 사용하여 이러한 문자열의 특정 부분을 추출한 다음 원하는 형식으로 연결합니다. 월의 경우 세 글자 약어를 추출한 다음 조회 문자열(monthMap)을 사용하여 숫자 해당 값을 찾습니다. 월이 한 자리 숫자이면 선행 0이 추가됩니다. 마지막으로 추출된 연도(마지막 두 자리), 월, 일, 시, 분 및 초를 연결하여 mpas_version 문자열을 형성합니다.

디스플레이 업데이트**:** 화면을 지우고 버전 정보를 중앙에 표시하여 버전 정보를 사용자에게 명확하게 제공합니다. 텍스트를 중앙에 배치하면 쉽게 읽을 수 있습니다. 기본 버전 문자열("MPAS-FE_Ver 0.1")과 컴파일 타임스탬프를 함께 표시하면 사람이 읽을 수 있는 버전과 더 정확한 빌드 식별자를 모두 제공합니다.

sprite.fillScreen(BLACK) 명령은 디스플레이를 지웁니다. 그런 다음 sprite.drawCentreString() 함수를 사용하여 화면에 수평으로 중앙에 두 줄의 텍스트를 그립니다. 첫 번째 줄은 기본 버전 문자열 "MPAS-FE_Ver 0.1"을 표시하고, 두 번째 줄은 밑줄 다음에 생성된 mpas_version 문자열을 표시합니다. 각 줄의 글꼴

크기도 drawCentreString() 함수에 지정됩니다.

주요 내용**:** 컴파일 날짜와 시간을 기준으로 펌웨어 버전을 표시합니다.

숨겨진 연결**:** 버전 정보의 정확도는 컴파일에 사용된 컴퓨터의 시스템 시계에 따라 달라집니다. 시스템 시계가 정확하지 않으면 버전 문자열도 정확하지 않습니다. 이 모드는 주로 정보 제공을 목적으로 하며 다른 기능과 직접적으로 상호 작용하지 않습니다.

광범위한 의미**:** 버전 정보를 제공하는 것은 소프트웨어 개발의 표준 관행으로, 사용자와 개발자가 다양한 릴리스를 추적하고 장치에서 현재 실행 중인 버전을 식별하는 데 도움이 됩니다. 이는 버그 보고, 기능 요청 및 호환성 보장에 매우 중요합니다.

4. 하드웨어 상호 작용 분석

- **4.1.** 디스플레이**:** M5.Display 및 M5Canvas(스프라이트) 객체를 통해 M5StickC Plus2의 LCD를 활용합니다. 모든 모드에서 부드러운 업데이트를 보장하고 깜박임을 방지하기 위해 스프라이트를 통한 더블 버퍼링을 사용합니다. 다양한 텍스트 크기,

색상 및 그리기 함수(fillRect, drawString, drawCentreString, qrcode)를 사용하여 각 모드에서 정보를 효과적으로 표시합니다. 일관되게 스프라이트 객체를 사용하여 모든 모드에서 더블 버퍼링을 수행하는 것은 임베디드 시스템의 디스플레이 최적화 기술에 대한 깊은 이해를 보여줍니다. 다양한 텍스트 서식 옵션을 통해 다양한 유형의 데이터를 명확하고 체계적으로 표시할 수 있습니다.

- **4.2.** 버튼 **(A** 및 **B):** 두 버튼 모두 모드 탐색 및 모드별 작업에 광범위하게 사용됩니다. 짧게, 중간, 길게 누르기를 구분하여 제한된 수의 버튼으로 더 풍부한 사용자 상호 작용을 제공합니다. 디바운싱은 M5.update() 함수에 의해 암시적으로 처리됩니다. 두 버튼 모두에서 다중 누르기 감지를 구현하면 사용자에게 액세스 가능한 기능이 최대화됩니다. 대부분의 모드에서 버튼 B를 모드 탐색에 일관되게 사용하면 예측 가능한 사용자 경험을 제공합니다.

- **4.3.** 마이크**:** 내장 마이크는 모터 주파수를 추정하기 위해 FFT를 통한 실시간 오디오 분석을 위해 모드 0 및 모드 1에서 사용됩니다. M5.Mic.record() 함수는 오디오 데이터를 캡처하는 데 사용됩니다. 코드는 모터 성능을 측정하기 위한 비접촉 센서로 마이크를 효과적으로 활용하며, 이는 의도된 애플리케이션의 핵심 기능입니다.

- **4.4. IMU (**관성 측정 장치**):** IMU는 방향 추정을 위해 가속도계 및 자이로스코프 데이터를 얻기 위해 모드 4 (수평계)에서 사용됩니다. M5.Imu.getAccelData() 및 M5.Imu.getGyroData() 함수는 센서 값을 읽는 데 사용됩니다. IMU 데이터를 통합하면 장치가 공간 인식을 제공하여 수평계 기능을 활성화할 수 있습니다. 칼만 필터와 같은 센서 융합 기술을 사용하면 정확하고 안정적인 방향 판독값을 얻으려는 노력을 보여줍니다.

- **4.5.** 전원 관리**:** 코드는 M5.Power.getBatteryVoltage()를 사용하여 배터리 전압을 얻고 M5.Power.setLed()를 사용하여 저전압 표시를 위해 LED를 제어하여 전원 관리 장치와 상호 작용합니다. 배터리 모니터링을 포함하는 것은 M5StickC Plus2와 같은 휴대용 장치에 매우 중요합니다. LED를 통한 저전압 경고는 사용자에게 장치를 충전해야 할 시기를 명확하게 시각적으로 알려줍니다.
- **4.6.** 부저**:** 코드는 initBuzzer()(buzzer.h에서)를 사용하여 부저를 초기화하지만, 제공된 loop() 함수에서는 부저가 명시적으로 사용되지 않습니다. 이는 부저 기능이 다른 곳에 구현되었거나 향후 사용을 위한 것임을 시사합니다(예: 버튼 누르기 또는 랩 목표 달성에 대한 청각적 피드백 제공). 부저를 초기화하면 사용자 경험을 향상시키기 위해 청각적 피드백을 추가할 수 있는 잠재력이 있음을 나타냅니다.

5. 신호 처리 측면

- **5.1. FFT** 구현**:** 코드는 고속 푸리에 변환을 수행하기 위해 arduinoFFT 라이브러리를 사용합니다. FFT는 SAMPLES = 512 및 SAMPLING_FREQUENCY = 700 Hz로 구성됩니다. 스펙트럼 누출을 줄이기 위해 FFT 계산 전에 해밍 윈도우가 입력

데이터에 적용됩니다. majorPeak() 함수는 스펙트럼에서 가장 큰 크기를 갖는 주파수를 찾는 데 사용됩니다. FFT 매개변수 선택은 주파수 분해능과 감지 가능한 최대 주파수 사이의 절충을 나타냅니다. 더 많은 샘플 수는 더 나은 주파수 분해능을 제공하지만 더 많은 처리 시간이 필요합니다. 700Hz의 샘플링 주파수는 분석을 350Hz(나이퀴스트 주파수) 미만의 주파수로 제한하며, 이는 미니 4WD 모터의 사운드를 분석하는 데 적합해 보입니다. 해밍 윈도우는 사이드 로브 레벨을 효과적으로 줄여 주요 주파수를 더 정확하게 추정하므로 스펙트럼 분석에 일반적으로 사용됩니다.

- **5.2.** 필터링 기술 **(**모드 **4):** 모드 4는 IMU 데이터를 처리하기 위해 여러 필터링 기술을 사용합니다.

    - 보상 필터**:** 별도의 함수로 명시적으로 표시되지는 않지만, 초기 접근 방식에는 가속도계와 자이로스코프 데이터를 융합하기 위한 보상 필터가 포함되었을 가능성이 높습니다. 칼만 필터로의 전환은 더 정교한 접근 방식으로의 발전을 시사합니다.

    - 저역 통과 필터 **(LPF):** #define ALPHA_LPF 0.01은 고주파 노이즈를 줄이기 위해 가속도계 데이터에 저역 통과 필터를 사용하려는 의도를 시사합니다. 그러나 칼만 필터 섹션에는 직접적으로 구현되지 않았습니다.

    - 지수 이동 평균 **(EMA):** #define EMA_ALPHA 0.2 및 applyEMAFilter() 함수는 각도 추정값을 스무딩하기 위해 지수 이동 평균 필터를 사용함을 나타냅니다. 그러나 이것도 최종 칼만 필터 구현에는 직접적으로 사용되지 않았습니다.

    - 칼만 필터**:** 모드 4의 각도 추정의 핵심은 칼만 필터입니다. 이 필터는 노이즈가 많은 가속도계 및 자이로스코프 측정값과 시스템의 동적 모델을 최적으로 결합하여 롤 및 피치 각도의 더 정확하고 안정적인 추정값을 제공합니다. 필터 매개변수(Q_angle, Q_bias, R_measure)는 필터의 동작을 제어하며 신중하게 조정해야 합니다. 더 간단한 필터(보상, LPF, EMA)에서 더 복잡한 칼만 필터로의 발전은 수평계 기능의 정확성과 견고성을 향상시키려는 노력을 나타냅니다. 칼만 필터는 노이즈 특성이 다른 여러 센서의 데이터를 융합하는 데 특히 적합합니다. 칼만 필터의 매개변수는 매우 중요하며 특정 M5StickC Plus2 IMU에 대해 최적의 성능을 얻기 위해 경험적 튜닝이 필요했을 가능성이 높습니다.

6. 수학적 및 알고리즘적 고려 사항

- **6.1.** 속도 계산 **(**모드 **0):** 속도는 공식 (rpmCurrent / selectedGear) * (PI * currentWheelDiameter * 0.06)을 사용하여 계산됩니다. 이 공식은 측정된 주파수에서 파생된 모터의 RPM을 가져와 기어비로 나누어 휠의 RPM을 얻은 다음 휠의 둘레와 변환 계수(0.06)를 곱하여 속도를 km/h로 얻습니다. 이 공식은 미니 4WD의 선형 속도를 추정하기 위해 회전 운동의 기본 물리학을 정확하게 적용합니다. 속도 추정의 정확도는 측정된 주파수, 사용자가 입력한 휠 직경 및 기어비, 휠 슬립이 없다는 가정에 따라 달라집니다. 변환 계수 0.06은 mm (휠 직경) 및 분 (RPM에서)에서 km 및 시간으로 필요한 단위 변환을 결합한 것 같습니다.

- **6.2.** 칼만 필터 방정식 **(**모드 **4):** kalmanFilter() 함수는 표준 칼만 필터 알고리즘을 구현합니다. 방정식에는 이전 상태와 제어 입력(자이로스코프 속도)을 기반으로 현재 상태(각도 및 바이어스)를 예측한 다음 최신 측정값(가속도계 각도)을 사용하여 이 예측을 업데이트하는 과정이 포함됩니다. 필터는 또한 상태 추정의 불확실성을 정량화하기 위해 오류 공분산 행렬(P_KF)의 추정값을 유지합니다. 칼만 필터는 측정값과 시스템 모델 모두의 불확실성을 고려하여 노이즈가 많은 센서 데이터를 최적으로 융합하는 방법을 제공합니다. 필터의 성능은 프로세스 노이즈 공분산(Q_angle, Q_bias)과 측정 노이즈 공분산(R_measure)의 올바른 튜닝에 크게 좌우됩니다. 이러한 매개변수는 사용된 IMU의 특정 특성을 기반으로 선택해야 합니다.

7. 사용자 인터페이스 및 사용자 경험

- **7.1.** 모드 탐색**:** 모드 전환은 주로 버튼 B를 짧게 및 중간 길이로 눌러 수행되며, 다양한 모드를 순환적으로 탐색할 수 있습니다. 다음 및 이전 모드로 탐색하기 위해 다양한 누르기 지속 시간을 사용하는 것은 단일 버튼으로 여러 기능을 관리하는 효율적인 방법입니다. 순환적인 특성은 모든 모드에 쉽게 액세스할 수 있도록 보장합니다.

- **7.2.** 정보 표시**:** 각 모드는 M5StickC Plus2의 화면에 관련 정보를 명확하게 표시하도록 설계되었습니다. 텍스트 크기, 색상 및 서식을 사용하여 주요 데이터를 강조 표시합니다. 스프라이트를 사용하여 더블 버퍼링을 수행하면 더 부드러운 시각적 경험에 기여합니다. 각 모드에서 시각적 표현을 신중하게 고려하면 사용자가 표시된 데이터를 더 잘 이해할 수 있습니다. 일부 모드(예: 랩 목표에 따른 스톱워치 타이머 색상)에서 색상 코딩을 사용하면 추가적인 시각적 단서를 제공합니다.

- **7.3.** 모드별 상호 작용**:** 각 모드에는 센서 모드에서 매개변수 구성, 스톱워치 모드에서 시작/랩/재설정, 수평계 모드에서 보정과 같이 버튼 A를 사용하는 고유한 특정 상호 작용이 있습니다. 버튼 A 동작을 각 모드의 특정 기능에 매핑하면 맞춤형 사용자 경험을 제공할 수 있습니다. 덜 자주 사용되는 동작(예: 최대 주파수 고정, 스톱워치 재설정, 수평계 보정)에 대해 길게 누르기를 사용하면 더 일반적인 작업에 대한 짧게 누르기 동작을 유지하는 데 도움이 됩니다.

- **7.4.** 피드백 메커니즘**:** 코드는 저전압 시 깜박이는 배터리 표시등 및 랩 목표 도달 시 스톱워치 모드에서 깜박이는 타이머와 같은 몇 가지 기본적인 피드백 메커니즘을 포함합니다. 부저(초기화되었지만 주 루프에서 사용되지 않음)를 사용하면 추가적인 청각적 피드백을 제공할 수 있습니다. 장치의 상태 및 사용자의 동작 결과에 대한 피드백을 사용자에게 제공하는 것은 좋은 사용자 경험에 매우 중요합니다. 기존의 시각적 피드백은 유용하며, 청각적 피드백을 추가하면 상호 작용이 더욱 향상될 수 있습니다.

8. 잠재적인 개선 사항 및 권장 사항

- **8.1.** 코드 가독성 및 유지 관리성**:** 특히 칼만 필터 구현 및 속도 계산 공식과 같이 복잡한 로직을 설명하기 위해 더 자세한 주석을 추가하는 것을 고려하십시오. 일부 변수의 이름을 더 설명적으로 변경하십시오(예: previousTimeCF를 lastImuUpdateTime으로). 자주 사용되는 상수(예: 속도 계산의 0.06)에 대해 의미 있는 이름을 정의하여 코드 이해도를 높이는 것을 고려하십시오.

- **8.2.** 성능 최적화**:** 모드 0 또는 모드 1에서 FFT 분석은 loop() 함수의 모든 반복에서 수행됩니다. 필요한 경우 FFT 처리 후 작은 지연을 추가하여 CPU 부하를 줄이는 것을 고려하십시오. 모드 4에서 칼만 필터 계산은 모든 IMU 업데이트에서 수행됩니다. IMU의 업데이트 속도는 수평계 기능에 필요한 것보다 높을 수 있습니다. 성능 또는 전력 소비가 문제가 되는 경우 IMU 데이터를 다운샘플링하거나 칼만 필터 업데이트 빈도를 조정하는 것을 고려하십시오.

- **8.3.** 사용자 인터페이스 개선 사항**:** 모드 0에서 명확성을 위해 최대 주파수(Hz) 및 현재 주파수(Hz)에 대한 단위를 표시하는 것을 고려하십시오. 스톱워치 모드에서 스톱워치가 실행 중인지 여부를 시각적으로 표시하는 것을 고려하십시오(예: 재생/일시 정지 아이콘). 버튼 누르기, 스톱워치에서 시작/정지/랩 이벤트 또는 랩 목표 도달에 대한 청각적 피드백을 제공하기 위해 부저를 사용하는 가능성을 탐색하십시오.

- **8.4.** 알고리즘 개선 사항**:** 모드 4에서 요 각도는 현재 자이로스코프 데이터의 단순 통합으로 추정되며, 이는 드리프트가 발생하기 쉽습니다. 자력계(M5StickC Plus2에 있는 경우) 또는 더 발전된 센서 융합 알고리즘을 사용하여 더 강력한 요 각도 추정 방법을 구현하는 것을 고려하십시오. 칼만 필터 매개변수는 하드코딩되어 있습니다. 구성 모드를 통해 또는 비휘발성 메모리에 저장하여 이러한 매개변수를 튜닝할 수 있는 메커니즘을 추가하는 것을 고려하십시오.

- **8.5.** 배터리 모니터링**:** getBatteryPercent() 함수에는 저전압 및 고전압에 대한 하드코딩된 값이 포함되어 있습니다. 이러한 값을 구성에서 읽거나 M5StickC Plus2의 사양을 기반으로 정의하는 것을 고려하십시오. battery_monitor.h 파일에는 더 고급 배터리 관리 기능이 포함될 수 있습니다. 이러한 기능을 탐색하고 잠재적으로 메인 코드에 통합하십시오.

- **8.6.** 오류 처리**:** 마이크 녹음이 성공했는지 또는 IMU 초기화가 성공했는지 확인하는 것과 같은 기본적인 오류 처리를 추가하십시오.

- **8.7.** 코드 모듈화**:** 각 모드의 로직을 별도의 함수로 이동하여 코드 가독성 및 유지 관리성을 향상시키는 것을 고려하십시오.

9. 결론

제공된 아두이노 코드는 *M5StickC Plus2*를 활용하여 미니 *4WD* 경주 애호가를 위한 다양한 기능을 갖춘 보조 장치를 성공적으로 구현 합니다.

이 프로젝트는 센서 데이터 수집, 실시간 분석 및 여러 유틸리티 모드를 통합하여 모터 성능 평가, 경주 시간 측정, 섀시 정렬, 배터리 모니터링 및 온라인 리소스 액세스를 위한 다재다능한 도구를 제공합니다. 디스플레이, 버튼, 마이크, IMU, 전원 관리와 같은 다양한 하드웨어 구성 요소와 FFT 및 칼만 필터와 같은 신호 처리 기술의 성공적인 통합이 입증되었습니다.

코드 가독성, 성능 최적화, 사용자 인터페이스 디자인 및 알고리즘 정교화와 같은 영역에서 추가 개선 및 향상의 잠재력이 여전히 존재합니다.

전반적으로 이 프로젝트는 **M5StickC Plus2**를 위한 잘 구조화되고 기능이 풍부한 애플리케이션으로 긍정적으로 평가할 수 있습니다**.**