Зертханалық жұмыс №3
EV датчиктерінің сигналдарын өңдеу және атқарушы құрылғыларды қауіпсіз басқару

1. Жұмыстың мақсаты

Электромобильді басқару жүйесіндегі датчиктердің аналогтық сигналдарын өңдеу, олардың рұқсат етілген жұмыс диапазонын тексеру, резервтелген датчик арналарының өзара сәйкестігін анықтау және ақау пайда болған кезде жүйені қауіпсіз күйге ауыстыру принциптерін зерттеу.

Зертханалық жұмыс барысында студент:

- бірнеше аналогтық датчик сигналдарын ADC арқылы оқуды;
- ADC мәндерін физикалық шамаға түрлендіруді;
- датчик сигналының жұмыс диапазонын тексеруді;
- Range Checking принципін қолдануды;
- резервтелген датчик арналарының жұмыс принципін түсінуді;
- Plausibility Checking орындауды;
- датчик сигналдарын қарапайым сүзгі арқылы өңдеуді;
- датчик ақауларын анықтауды;
- Fault Detection алгоритмін құруды;
- Safe State принципін түсінуді;
- ақау кезінде атқарушы құрылғыны қауіпсіз түрде ажыратуды;

2. Алдыңғы зертханалық жұмыстармен байланысы

№1 зертханалық жұмыста:
GPIO + ADC + датчик сигналдарын оқу + физикалық шамаға түрлендіру қарастырылды.

№2 зертханалық жұмыста:
Hardware Timer → Interrupt → Periodic Tasks → PWM
қарастырылды.

№3 зертханалық жұмыста негізгі сұрақ:
Микроконтроллер датчиктен алынған сигналдың дұрыс екенін қалай анықтайды және ақау анықталған кезде не істеуі тиіс?

Жалпы логика:
Датчик → ADC → Signal Processing → Range Checking → Plausibility Checking → Sensor Valid? → Fault Detection → Safe Control

3. Датчик сигналдарын тексеру не үшін қажет?
Электромобильді басқару жүйесінде көптеген шешімдер датчиктерден алынған ақпаратқа байланысты болады.
Мысалы:

- акселератор педалінің орналасуы;
- тежеу педалінің орналасуы;
- батарея температурасы;
- батарея кернеуі;
- батарея тогы;
- инвертор температурасы;
- электр қозғалтқышының температурасы;
- салқындату жүйесінің қысымы.

Датчик немесе оның электрлік тізбегі зақымданса, микроконтроллер қате мән алуы мүмкін.

Мысалы: Нақты педаль күйі = 20 %
Бірақ датчик сигналы: Pedal = 90 % → Қате басқару сұранысы

Сондықтан автомобильдік басқару жүйесінде датчик сигналы:

оқылады → өңделеді → тексеріледі → содан кейін ғана

“басқару алгоритмінде қолданылады”

4. Қолданылатын бағдарламалық орта
Зертханалық жұмыста:

- Wokwi онлайн симуляторы;
- STM32F103C8T6 Blue Pill;
- Wokwi Code Editor;
- Serial Monitor

қолданылады.
Қосымша бағдарламаларды орнату қажет емес.
№1 және №2 зертханалық жұмыстарында қолданылған STM32F103C8T6 жобасының көшірмесін пайдалануға болады.

Немесе жаңадан жоба:
https://wokwi.com/projects/476622661513474049
Жобаның көшірмесін жасап, оны келесідей атаңыз:
Lab_03_Sensor_Safety_СіздіңАтыңыз

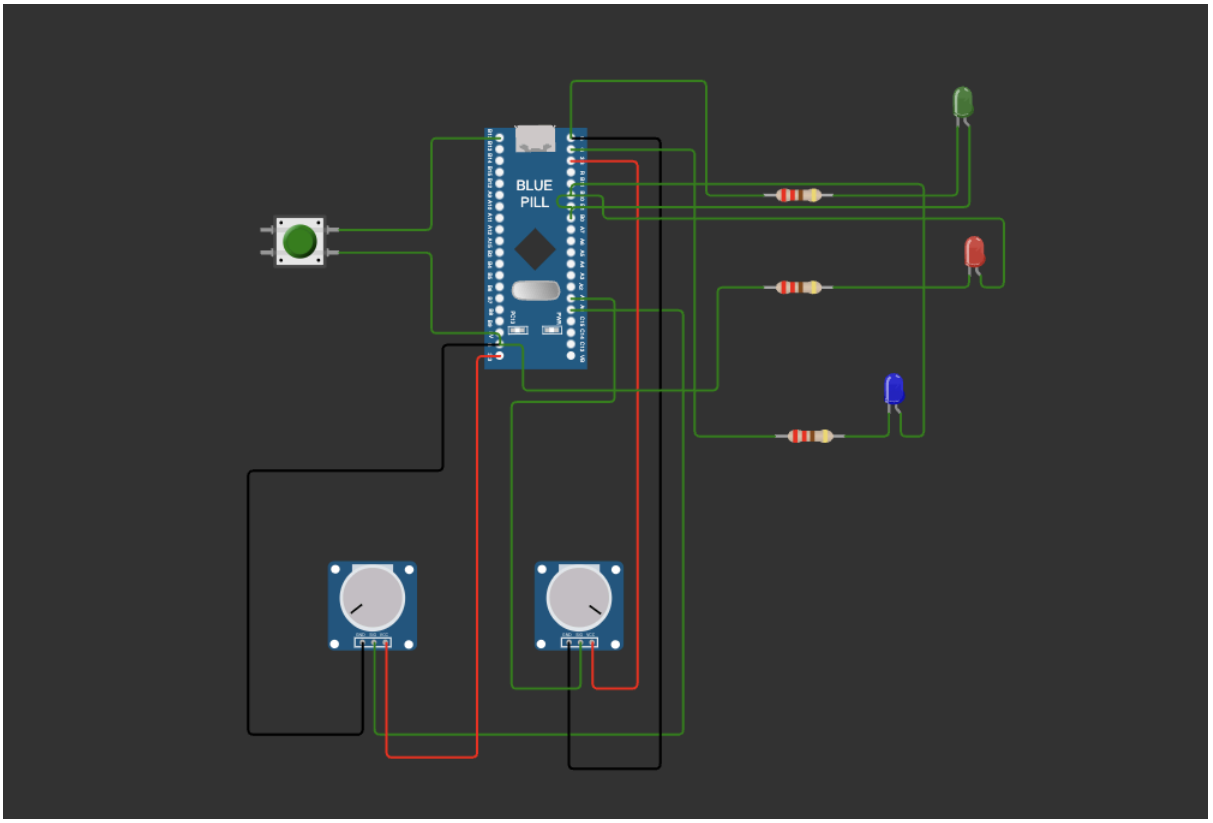
5. Қажетті компоненттер

Компонент	Саны	Мақсаты
STM32F103C8T6 Blue Pill	1	микроконтроллер
Potentiometer	2	аналогтық датчиктерді модельдеу
Switch немесе Button	1	ENABLE сигналы
READY LED	1	жүйенің қалыпты жұмыс күйі
FAULT LED	1	ақау күйі
ACTUATOR/CONTACTOR LED	1	атқарушы құрылғының Enable сигналын модельдеу
220–330 Ω резистор	3	LED тогын шектеу

6. STM32 кірістері мен шығыстары

Элемент	STM32
Potentiometer №1	PA0
Potentiometer №2	PA1
ENABLE	PB12
READY LED	PB0
FAULT LED	PB1
ACTUATOR / CONTACTOR LED	PB10

Жалпы схема:



7. Датчиктің жұмыс диапазоны

Жалпы ақпарат

STM32F103 микроконтроллерінде 12-биттік ADC қолданылған кезде теориялық ADC диапазоны:

0 ... 4095

Датчиктің қалыпты сигналын міндетті түрде осы диапазонның барлығын пайдаланбауы мүмкін.

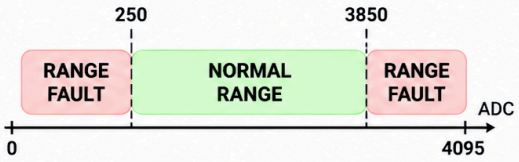
Зертханалық жұмыстың мақсаты

Датчиктің нақты жұмыс диапазонын (Range) анықтап, сигналды тексеру (Range Checking) принципін зерттеу.

Зерттеуде қолданылатын мәндер

ADC_MIN	250
ADC_MAX	3850

Сигнал диапазоны



Диапазонды тексеру логикасы

Шарт	Нәтиже
ADC < 250	**RANGE FAULT**
250 ≤ ADC ≤ 3850	**NORMAL RANGE**
ADC > 3850	**RANGE FAULT**

Кернеуге сәйкестік (V_ref = 3.3 В)

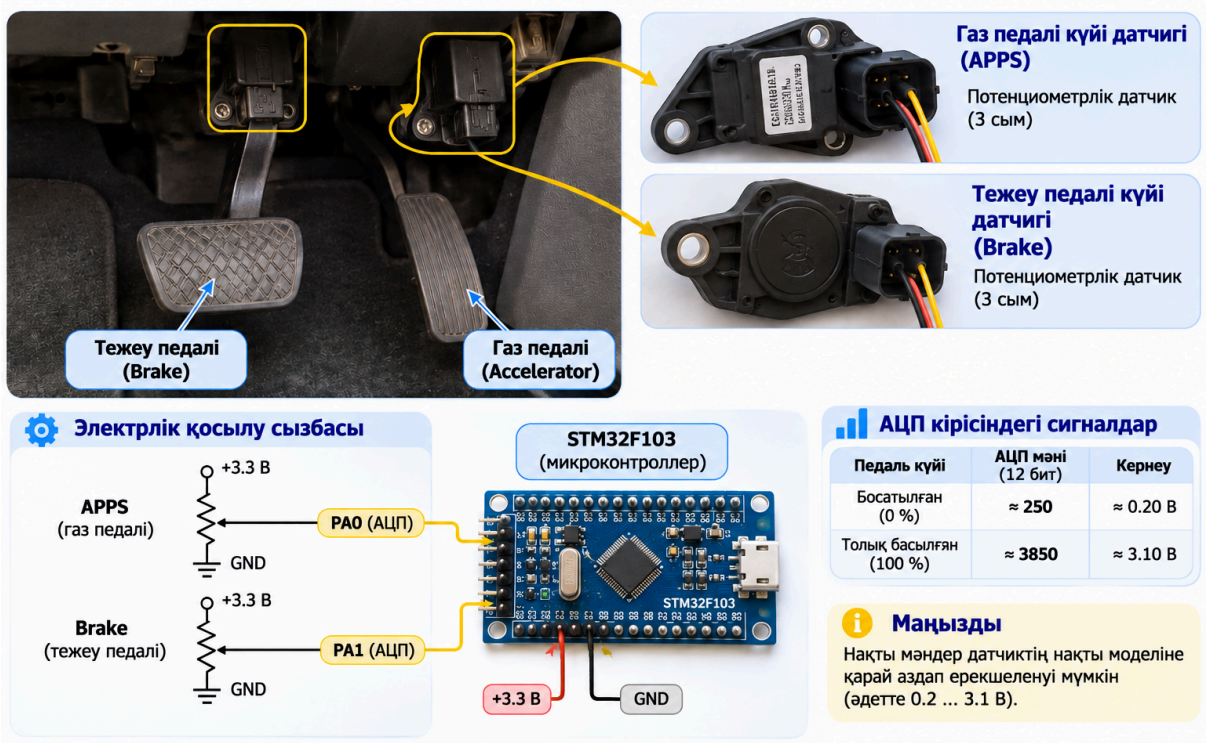
ADC	Кернеу, U
250	≈ 0.20 В
3850	≈ 3.10 В

Қорытынды

- Датчиктің нақты жұмыс диапазоны: 250 … 3850 (ADC).
- Бұл кезең нақты автомобиль датчигінің техникалық сипаттамасы емес, диапазонды тексеру принципін зерттеу үшін алған шартты мәндер.

1-тапсырма. Акселератор және тежеу педальдарының сигналдарын оқу

Алғашқы тапсырмада екі потенциометр екі әртүрлі датчикті модельдейді.



Педаль күйі	АЦП мәні (12 бит)	Кернеу
Босатылған (0 %)	≈ 250	≈ 0.20 В
Толық басылған (100 %)	≈ 3850	≈ 3.10 В

1-тапсырманың толық коды

```
#define ACCEL_PIN PA0
#define BRAKE_PIN PA1


// ==========================================
// ДАТЧИКТІҢ ЖҰМЫС ДИАПАЗОНЫ
// ==========================================

const int ADC_MIN = 250;
const int ADC_MAX = 3850;


// ==========================================
// ADC → 0–100 %
// ==========================================
```

```
float adcToPercent(int adc)
{
  if (adc <= ADC_MIN)
  {
    return 0.0;
  }

  if (adc >= ADC_MAX)
  {
    return 100.0;
  }

  return
    (adc - ADC_MIN)
    * 100.0
    / (ADC_MAX - ADC_MIN);
}


// ==========================================
// SETUP
// ==========================================

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

  // STM32F103 ADC = 12 bit
  analogReadResolution(12);
}


// ==========================================
// LOOP
// ==========================================

void loop()
{
  // Акселератор ADC
  int adcAccelerator =
    analogRead(ACCEL_PIN);

  // Тежеу педалі ADC
```

```
  int adcBrake =
    analogRead(BRAKE_PIN);


  // ADC → %
  float acceleratorPercent =
    adcToPercent(adcAccelerator);

  float brakePercent =
    adcToPercent(adcBrake);


  // ========================================
  // SERIAL MONITOR
  // ========================================

  Serial.print("ACCELERATOR ADC = ");
  Serial.print(adcAccelerator);

  Serial.print(" | Accelerator = ");
  Serial.print(acceleratorPercent, 1);
  Serial.print(" %");


  Serial.print(" || BRAKE ADC = ");
  Serial.print(adcBrake);

  Serial.print(" | Brake = ");
  Serial.print(brakePercent, 1);
  Serial.println(" %");


  delay(500);
}
```

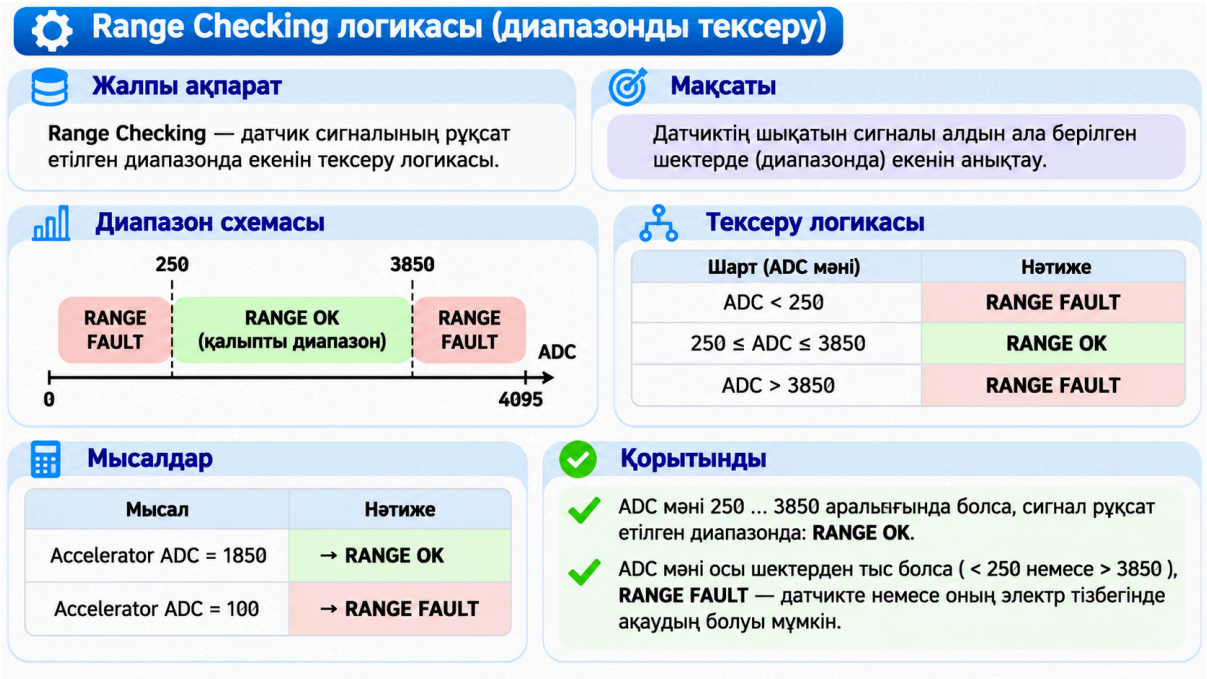
2-тапсырма. Датчик сигналдарының жұмыс диапазонын тексеру — Range Checking

Алдыңғы тапсырмада микроконтроллер екі датчиктің сигналын оқыды:

Accelerator Sensor + Brake Sensor

Енді әр датчик сигналының рұқсат етілген электрлік диапазонда орналасуын тексереміз.

Range Checking негізгі сұрағы: Датчиктің сигналы физикалық тұрғыдан рұқсат етілген диапазонда орналасқан ба?



ADC мәні физикалық тұрғыдан 0–4095 аралығында болуы мүмкін, бірақ бұл оның датчиктің рұқсат етілген сигналы екенін білдірмейді. Біздің оқу моделімізде тек 250–3850 аралығын қалыпты деп алдық. 2-тапсырмада біз датчик сигналының рұқсат етілген жұмыс диапазонын арнайы енгіздік.
Бұл тек бағдарламалау үшін ғана қажет емес — бұл датчиктің немесе электр сымдарының ақауын анықтау тәсілдерінің бірі. Нақты акселератор педалінің датчигін қарастырайық. Контроллер одан аналогтық кернеу сигналын қабылдайды. Автомобиль жүйелерінде қоректендіру кернеуінің төменгі және жоғарғы шекараларына жақын аймақтарда әдейі белгілі бір қор (специально оставляют запас) қалдырылады.

Яғни, қалыпты жұмыс істейтін датчик акселератор педалі толық босатылған кезде міндетті түрде дәл 0 В, ал педаль толық басылған кезде міндетті түрде ең жоғары кернеу мәнін беруі шарт емес.



2-тапсырманың толық коды

```
#define ACCEL_PIN PA0
#define BRAKE_PIN PA1


// ==========================================
// ДАТЧИКТІҢ ЖҰМЫС ДИАПАЗОНЫ
// ==========================================

const int ADC_MIN = 250;
const int ADC_MAX = 3850;


// ==========================================
// ADC → %
// ==========================================

float adcToPercent(int adc)
{
  if (adc <= ADC_MIN)
  {
    return 0.0;
  }
```

```
  if (adc >= ADC_MAX)
  {
    return 100.0;
  }

  return
    (adc - ADC_MIN)
    * 100.0
    / (ADC_MAX - ADC_MIN);
}


// ========================================
// RANGE CHECKING
// ========================================

bool checkRange(int adc)
{
  if (adc < ADC_MIN)
  {
    return false;
  }

  if (adc > ADC_MAX)
  {
    return false;
  }

  return true;
}


// ========================================
// SETUP
// ========================================

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

  analogReadResolution(12);
}
```

```
// ========================================
// LOOP
// ========================================

void loop()
{
  // ADC мәндерін оқу
  int adcAccelerator =
    analogRead(ACCEL_PIN);

  int adcBrake =
    analogRead(BRAKE_PIN);


  // ADC → %
  float acceleratorPercent =
    adcToPercent(adcAccelerator);

  float brakePercent =
    adcToPercent(adcBrake);


  // Range Checking
  bool acceleratorRangeOK =
    checkRange(adcAccelerator);

  bool brakeRangeOK =
    checkRange(adcBrake);


  Serial.println(
    "==================================="
  );


  // ACCELERATOR
  Serial.print("ACCELERATOR ADC = ");
  Serial.print(adcAccelerator);

  Serial.print(" | Accelerator = ");
  Serial.print(acceleratorPercent, 1);
```

```
  Serial.print(" %");

  Serial.print(" | RANGE = ");

  if (acceleratorRangeOK)
  {
    Serial.println("OK");
  }
  else
  {
    Serial.println("FAULT");
  }


  // BRAKE
  Serial.print("BRAKE ADC = ");
  Serial.print(adcBrake);

  Serial.print(" | Brake = ");
  Serial.print(brakePercent, 1);
  Serial.print(" %");

  Serial.print(" | RANGE = ");

  if (brakeRangeOK)
  {
    Serial.println("OK");
  }
  else
  {
    Serial.println("FAULT");
  }


  Serial.println(
    "==================================="
  );

  Serial.println();


  delay(500);
}
```

```
================================
ACCELERATOR ADC = 120 | Accelerator = 0.0 % | RANGE = FAULT
BRAKE ADC = 3747 | Brake = 97.1 % | RANGE = OK
================================
```

Яғни, Range Checking (мәннің рұқсат етілген диапазонда болуын тексеру) келесі ақауларды анықтауға көмектеседі:

- сигнал деңгейінің тым төмен болуы;
- сигнал деңгейінің тым жоғары болуы;
- жерге (GND) қысқа тұйықталу;
- қоректендіру көзіне қысқа тұйықталу;
- электр тізбегінің үзілуінің немесе ақауының кейбір түрлері;
- датчик қалыпты жұмыс істеген кезде физикалық тұрғыдан пайда болмауы тиіс мәндердің анықталуы.

3-тапсырма. Акселератор педалінің екі арнасын салыстыру — Plausibility Checking

Бұл тапсырмада екі потенциометрдің қызметі өзгереді.
Енді олар бір акселератор педалінің екі сигналдық арнасын модельдейді:

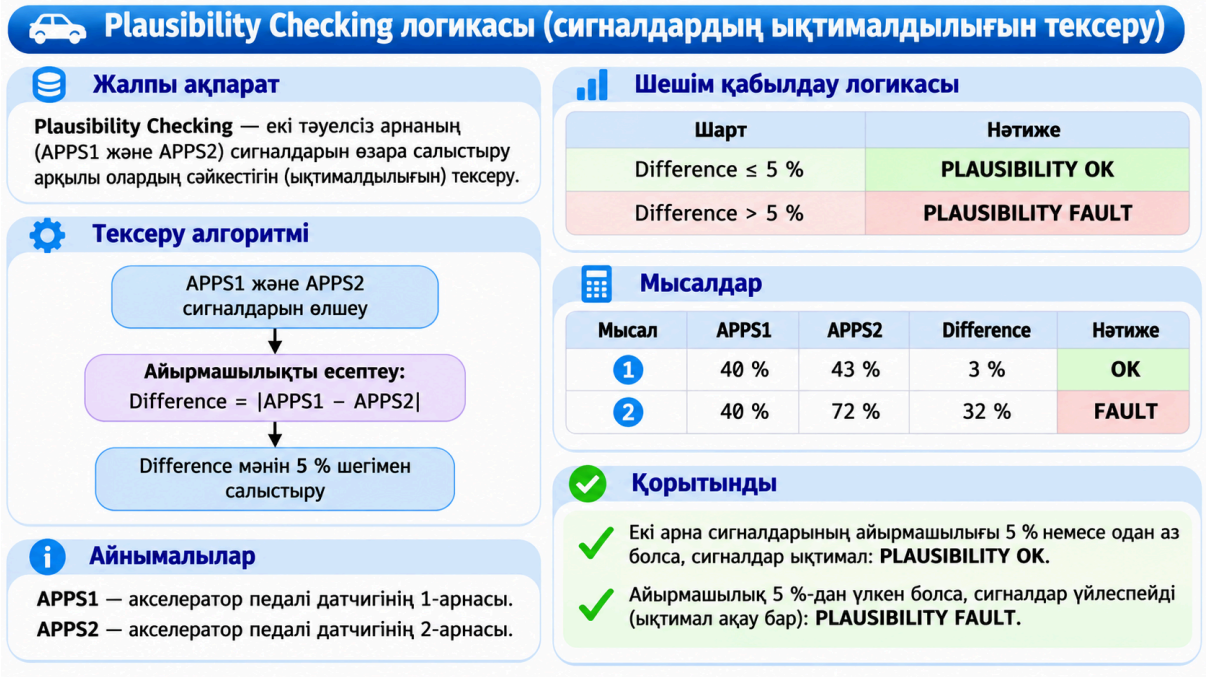


Неліктен екі арна қажет?

Маңызды автомобиль датчиктерінде бір физикалық шаманы анықтау үшін екі тәуелсіз немесе өзара салыстырылатын сигнал қолданылуы мүмкін.

Микроконтроллер: APPS1 + APPS2 → Салыстыру → Сигналдар сәйкес пе?

деп тексереді.



3-тапсырманың толық коды

```
#include <math.h>

#define APPS1_PIN PA0
#define APPS2_PIN PA1


const int ADC_MIN = 250;
const int ADC_MAX = 3850;

const float MAX_DIFFERENCE = 5.0;


// ==========================================
// ADC → %
// ==========================================

float adcToPercent(int adc)
{
  if (adc <= ADC_MIN)
  {
    return 0.0;
  }

  if (adc >= ADC_MAX)
```

```
  {
    return 100.0;
  }

  return
    (adc - ADC_MIN)
    * 100.0
    / (ADC_MAX - ADC_MIN);
}


// ========================================
// RANGE CHECK
// ========================================

bool checkRange(int adc)
{
  return
    (adc >= ADC_MIN)
    &&
    (adc <= ADC_MAX);
}


// ========================================
// SETUP
// ========================================

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

  analogReadResolution(12);
}


// ========================================
// LOOP
// ========================================

void loop()
{
  int adc1 =
```

```
  analogRead(APPS1_PIN);

int adc2 =
  analogRead(APPS2_PIN);


bool apps1RangeOK =
  checkRange(adc1);

bool apps2RangeOK =
  checkRange(adc2);


float apps1 =
  adcToPercent(adc1);

float apps2 =
  adcToPercent(adc2);


float difference =
  fabs(apps1 - apps2);


bool plausibilityOK =
  apps1RangeOK
  &&
  apps2RangeOK
  &&
  (difference <= MAX_DIFFERENCE);


Serial.println(
  "==================================="
);


Serial.print("APPS1 = ");
Serial.print(apps1, 1);
Serial.print(" %");

Serial.print(" | RANGE = ");
Serial.println(
```

```
    apps1RangeOK ? "OK" : "FAULT"
  );


  Serial.print("APPS2 = ");
  Serial.print(apps2, 1);
  Serial.print(" %");

  Serial.print(" | RANGE = ");
  Serial.println(
    apps2RangeOK ? "OK" : "FAULT"
  );


  Serial.print("Difference = ");
  Serial.print(difference, 1);
  Serial.println(" %");


  Serial.print("PLAUSIBILITY = ");

  if (plausibilityOK)
  {
    Serial.println("OK");
  }
  else
  {
    Serial.println("FAULT");
  }


  Serial.println(
    "================================="
  );

  Serial.println();


  delay(500);
}
```

```
==================================
APPS1 = 54.6 % | RANGE = OK
APPS2 = 57.9 % | RANGE = OK
Difference = 3.3 %
PLAUSIBILITY = OK
==================================
```

! 5 % және 6 % айырмашылықтарын міндетті түрде тексеріңіз.

4-тапсырма. Датчик сигналдарын сүзгілеу — Filtering

Сигналдарды сүзгілеу (Signal Filtering) қажет, өйткені тіпті дұрыс жұмыс істейтін датчиктің өзі аздап өзгеріп тұратын мәндерді беруі мүмкін. Бұл датчиктің ақаулы екенін білдірмейді. Көбінесе мұның себебі электрлік кедергілер, діріл және микроконтроллердің сигналды өлшеу ерекшеліктері болып табылады.

Мұны электромобильдің акселератор педалі мысалында қарастырайық. Жүргізуші акселератор педалін 50%-ға басып, оны қозғалтпай ұстап тұр делік.

Идеалды жағдайда микроконтроллер ADC арқылы үнемі бірдей мән алуы керек:

2000 → 2000 → 2000 → 2000 → 2000

Бірақ нақты жағдайда мынадай мәндер алынуы мүмкін:

2000 → 2008 → 1995 → 2004 → 1998

Жүргізуші педальді мүлдем қозғалтпаса да, ADC мәндері аздап өзгеріп тұр.

Осындай шағын ауытқулар шу (Noise) әсерінен пайда болуы мүмкін.

Мұндағы шу — біз еститін дыбыс емес. Бұл электрлік сигналдың қажетсіз ауытқуы.

Шу неліктен пайда болады?
Автомобильде электрлік кедергілердің бірнеше көзі бар.

Шу көзі	Сигналға әсері
Электрқозғалтқыш және инвертор	Күштік транзисторлардың жылдам ауысып-қосылуы электромагниттік кедергілер тудырады
DC/DC түрлендіргіші	Кернеудің пульсациясын және электрлік кедергілерді тудыруы мүмкін
Электр сымдары	Жақын орналасқан электр тізбектерінен электромагниттік кедергілерді қабылдауы мүмкін
Автомобильдің дірілі	Механикалық өзгерістерге және электрлік контактілердің тұрақсыздығына әсер етуі мүмкін
Микроконтроллердің ADC модулі	Өлшеу кезінде өзіндік шу мен қателіктерге ие
Датчиктің қоректендіру кернеуі	Кернеудің аздаған өзгерістері өлшенетін сигналға әсер етуі мүмкін

Қазіргі акселератор педальдерінде көбінесе контактісіз Холл датчиктері қолданылады. Бірақ мұндай датчиктер де электрлік кедергілерден толық қорғалмаған.

Сигналды сүзгілемесек не болады?
Микроконтроллер акселератор педалінің орналасуына байланысты электрқозғалтқыштың қажетті айналу моментін есептейді.
Жүргізуші педальді 50% деңгейінде ұстап тұр
ADC — сигналы 2000 → 2008 → 1995 → 2004 сүзгілеусіз аздап ауытқиды
Өлшенген мәндердің шағын ауытқулары айналу моментін есептеуге әсер етуі мүмкін. Басқару алгоритміне байланысты бұл қозғалтқышқа берілетін команданың қажетсіз өзгеруіне әкелуі ықтимал.

Ал Сүзгілеумен – Басқару командасы тұрақтырақ болады

Сигналдарды сүзгілеу қалай жұмыс істейді?
Ең қарапайым әдістердің бірі — жылжымалы орташа мән әдісі (Moving Average).
Микроконтроллер ADC арқылы төрт мән алды делік:

$$ADC_{filtered} = \frac{2010+1995+2004+1999}{4}$$

ADCfiltered = 2002

Яғни, бағдарлама әрбір шағын ауытқуға жеке әрекет етудің орнына, сүзгіленген орташа мәнді пайдаланады.

Бірақ мұнда маңызды мәселе бар!

Егер жүргізуші акселератор педалін шынымен жылдам басса ше?
Мысалы: 1000 → 1500 → 2500 → 3500
Бұл — шу емес! Бұл педаль орналасуының нақты өзгеруі.
Егер сүзгілеу тым күшті болса, микроконтроллер жүргізушінің әрекетіне кеш жауап беруі мүмкін.

Сондықтан автомобиль жүйелерінде мынадай талаптарды сақтау маңызды:
- Кездейсоқ шағын ауытқуларды азайту.
- Жүргізушінің нақты әрекеттеріне жылдам жауап беру.
- Датчиктің қауіпті ақауларын жасырмау.

Нақты автомобиль жүйелерінде сүзгілеу алгоритмдері қауіпсіздік талаптары мен жауап беру уақыты ескеріле отырып жобаланады.

Range Checking — датчик сигналының рұқсат етілген диапазонда екенін тексеру.
Signal Filtering — өлшеу сигналындағы шуды және қажетсіз ауытқуларды азайту.
Plausibility Checking — акселератор педалінің екі арнасынан алынған сигналдардың өзара сәйкестігін тексеру.

4-тапсырманың толық коды

```
#include <Arduino.h>
#include <math.h>


// ==========================================
// APPS КІРІСТЕРІ
// ==========================================

#define APPS1_PIN PA0
#define APPS2_PIN PA1


// ==========================================
// ADC ЖҰМЫС ДИАПАЗОНЫ
// ==========================================

const int ADC_MIN = 250;
const int ADC_MAX = 3850;


// ==========================================
// PLAUSIBILITY ШЕГІ
// ==========================================

const float MAX_DIFFERENCE = 5.0;


// ==========================================
// MOVING AVERAGE FILTER
// ==========================================

// Соңғы 5 өлшеуді қолданамыз

const int FILTER_SIZE = 5;


// APPS1 және APPS2 үшін буферлер

int apps1Buffer[FILTER_SIZE] =
```

```
{
  0, 0, 0, 0, 0
};

int apps2Buffer[FILTER_SIZE] =
{
  0, 0, 0, 0, 0
};


// Өлшеулер қосындысы

long apps1Sum = 0;
long apps2Sum = 0;


// Буфердегі ағымдағы позиция

int filterIndex = 0;


// Буферге қанша нақты өлшеу енгізілді

int filterCount = 0;


// Сүзгіден кейінгі ADC мәндері

float apps1FilteredADC = 0.0;
float apps2FilteredADC = 0.0;


// ========================================
// ADC → 0–100 %
// ========================================

float adcToPercent(float adc)
{
  if (adc <= ADC_MIN)
  {
    return 0.0;
  }
```

```
  if (adc >= ADC_MAX)
  {
    return 100.0;
  }

  return
    (adc - ADC_MIN)
    * 100.0
    / (ADC_MAX - ADC_MIN);
}


// ========================================
// RANGE CHECKING
// ========================================

bool checkRange(int adc)
{
  return
    (adc >= ADC_MIN)
    &&
    (adc <= ADC_MAX);
}


// ========================================
// MOVING AVERAGE FILTER
// ========================================

void updateFilter(
  int raw1,
  int raw2
)
{
  // Буфердегі ескі мәндерді
  // жалпы қосындыдан алып тастаймыз

  apps1Sum -=
    apps1Buffer[filterIndex];

  apps2Sum -=
    apps2Buffer[filterIndex];
```

```
// Жаңа ADC мәндерін буферге жазамыз

apps1Buffer[filterIndex] =
  raw1;

apps2Buffer[filterIndex] =
  raw2;


// Жаңа мәндерді қосындыға қосамыз

apps1Sum += raw1;
apps2Sum += raw2;


// Келесі буфер ұяшығына өтеміз

filterIndex++;

if (filterIndex >= FILTER_SIZE)
{
  filterIndex = 0;
}


// Алғашқы іске қосылған кезде
// буфер бірден 5 мәнге толмайды

if (filterCount < FILTER_SIZE)
{
  filterCount++;
}


// Орташа ADC мәндерін есептеу

apps1FilteredADC =
  apps1Sum
  / (float)filterCount;

apps2FilteredADC =
  apps2Sum
```

```
    / (float)filterCount;
}


// ================================================
// SETUP
// ================================================

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

  // STM32F103 ADC = 12 bit
  // 0 ... 4095

  analogReadResolution(12);
}


// ================================================
// LOOP
// ================================================

void loop()
{
  // ========================================
  // 1. RAW ADC МӘНДЕРІН ОҚУ
  // ========================================

  int adc1 =
    analogRead(APPS1_PIN);

  int adc2 =
    analogRead(APPS2_PIN);


  // ========================================
  // 2. RANGE CHECKING
  // ========================================

  bool apps1RangeOK =
    checkRange(adc1);
```

```
bool apps2RangeOK =
  checkRange(adc2);


// ========================================
// 3. FILTERING
// ========================================

// Сүзгіге тек рұқсат етілген
// сигналдарды жібереміз

if (
  apps1RangeOK
  &&
  apps2RangeOK
)
{
  updateFilter(
    adc1,
    adc2
  );
}


// ========================================
// 4. FILTERED ADC → %
// ========================================

float apps1Percent =
  adcToPercent(
    apps1FilteredADC
  );

float apps2Percent =
  adcToPercent(
    apps2FilteredADC
  );


// ========================================
// 5. PLAUSIBILITY CHECKING
// ========================================
```

```
float difference =
  fabs(
    apps1Percent
    -
    apps2Percent
  );


bool plausibilityOK =
  apps1RangeOK
  &&
  apps2RangeOK
  &&
  (difference <= MAX_DIFFERENCE);


// ========================================
// 6. SERIAL MONITOR
// ========================================

Serial.println(
  "============================================="
);


// ---------- APPS1 ----------

Serial.print("APPS1 RAW ADC = ");
Serial.println(adc1);

Serial.print("APPS1 FILTERED ADC = ");
Serial.println(
  apps1FilteredADC,
  1
);

Serial.print("APPS1 = ");
Serial.print(
  apps1Percent,
  1
);

Serial.println(" %");
```

```
Serial.print("APPS1 RANGE = ");

Serial.println(
  apps1RangeOK
    ? "OK"
    : "FAULT"
);


Serial.println();


// ---------- APPS2 ----------

Serial.print("APPS2 RAW ADC = ");
Serial.println(adc2);

Serial.print("APPS2 FILTERED ADC = ");
Serial.println(
  apps2FilteredADC,
  1
);

Serial.print("APPS2 = ");
Serial.print(
  apps2Percent,
  1
);

Serial.println(" %");


Serial.print("APPS2 RANGE = ");

Serial.println(
  apps2RangeOK
    ? "OK"
    : "FAULT"
);
```

```
  Serial.println();


  // ---------- PLAUSIBILITY ----------

  Serial.print("Difference = ");

  Serial.print(
    difference,
    1
  );

  Serial.println(" %");


  Serial.print("PLAUSIBILITY = ");

  Serial.println(
    plausibilityOK
      ? "OK"
      : "FAULT"
  );


  Serial.println(
    "=========================================="
  );

  Serial.println();


  delay(500);
}
```

```
APPS1 RAW ADC = 1957
APPS1 FILTERED ADC = 1957.0
APPS1 = 47.4 %
APPS1 RANGE = OK

APPS2 RAW ADC = 1033
APPS2 FILTERED ADC = 1022.4
APPS2 = 21.5 %
APPS2 RANGE = OK

Difference = 26.0 %
PLAUSIBILITY = FAULT
```

5-тапсырма. Fault Detection және Safe State



5-тапсырманың толық коды

```
#include <Arduino.h>
#include <math.h>


// ==========================================
// APPS
// ==========================================

#define APPS1_PIN PA0
#define APPS2_PIN PA1


// ==========================================
// DIGITAL INPUT
// ==========================================

#define ENABLE_PIN PB12


// ==========================================
// OUTPUTS
// ==========================================

#define READY_LED PB0
#define FAULT_LED PB1
#define ACTUATOR_LED PB10


// ==========================================
// ADC RANGE
// ==========================================

const int ADC_MIN = 250;
const int ADC_MAX = 3850;


// ==========================================
// PLAUSIBILITY
// ==========================================

const float MAX_DIFFERENCE = 5.0;
```

```
// ==========================================
// FILTER
// ==========================================

const int FILTER_SIZE = 5;

int apps1Buffer[FILTER_SIZE] = {0};
int apps2Buffer[FILTER_SIZE] = {0};

long apps1Sum = 0;
long apps2Sum = 0;

int filterIndex = 0;
int filterCount = 0;

float apps1FilteredADC = 0;
float apps2FilteredADC = 0;


// ==========================================
// TIMER
// ==========================================

HardwareTimer *ControlTimer =
  new HardwareTimer(TIM2);

volatile bool task100ms = false;


// ==========================================
// VARIABLES
// ==========================================

int adc1 = 0;
int adc2 = 0;

float apps1Percent = 0;
float apps2Percent = 0;

float difference = 0;

float controlRequest = 0;
```

```
bool apps1RangeOK = false;
bool apps2RangeOK = false;

bool plausibilityOK = false;
bool sensorValid = false;

const char *systemState = "OFF";

int diagnosticCounter = 0;


// ==========================================
// TIMER ISR
// ==========================================

void timerISR()
{
  task100ms = true;
}


// ==========================================
// ADC → %
// ==========================================

float adcToPercent(float adc)
{
  if (adc <= ADC_MIN)
  {
    return 0.0;
  }

  if (adc >= ADC_MAX)
  {
    return 100.0;
  }

  return
    (adc - ADC_MIN)
    * 100.0
    / (ADC_MAX - ADC_MIN);
}
```

```
// ========================================
// RANGE CHECK
// ========================================

bool checkRange(int adc)
{
  return
    (adc >= ADC_MIN)
    &&
    (adc <= ADC_MAX);
}


// ========================================
// FILTER
// ========================================

void updateFilter(
  int raw1,
  int raw2
)
{
  apps1Sum -=
    apps1Buffer[filterIndex];

  apps2Sum -=
    apps2Buffer[filterIndex];


  apps1Buffer[filterIndex] =
    raw1;

  apps2Buffer[filterIndex] =
    raw2;


  apps1Sum += raw1;
  apps2Sum += raw2;


  filterIndex++;
```

```
  if (filterIndex >= FILTER_SIZE)
  {
    filterIndex = 0;
  }


  if (filterCount < FILTER_SIZE)
  {
    filterCount++;
  }


  apps1FilteredADC =
    apps1Sum
    / (float)filterCount;

  apps2FilteredADC =
    apps2Sum
    / (float)filterCount;
}


// ========================================
// SENSOR PROCESSING
// ========================================

void processSensors()
{
  // RAW ADC
  adc1 =
    analogRead(APPS1_PIN);

  adc2 =
    analogRead(APPS2_PIN);


  // Range Checking RAW ADC арқылы орындалады
  apps1RangeOK =
    checkRange(adc1);

  apps2RangeOK =
    checkRange(adc2);
```

```
// Егер екі сигнал да электрлік диапазонда болса,
// сүзгіні жаңартамыз

if (
  apps1RangeOK
  &&
  apps2RangeOK
)
{
  updateFilter(
    adc1,
    adc2
  );
}


// Filtered ADC → %
apps1Percent =
  adcToPercent(
    apps1FilteredADC
  );

apps2Percent =
  adcToPercent(
    apps2FilteredADC
  );


// Difference
difference =
  fabs(
    apps1Percent
    -
    apps2Percent
  );


plausibilityOK =
  apps1RangeOK
  &&
  apps2RangeOK
```

```
    &&
    (difference <= MAX_DIFFERENCE);


  sensorValid =
    apps1RangeOK
    &&
    apps2RangeOK
    &&
    plausibilityOK;
}


// ========================================
// SAFE CONTROL
// ========================================

void updateControl()
{
  bool enable =
    (digitalRead(ENABLE_PIN) == LOW);


  // SYSTEM OFF
  if (!enable)
  {
    systemState = "OFF";

    controlRequest = 0;


    digitalWrite(
      READY_LED,
      LOW
    );

    digitalWrite(
      FAULT_LED,
      LOW
    );

    digitalWrite(
      ACTUATOR_LED,
```

```
    LOW
  );

  return;
}


// SENSOR FAULT
if (!sensorValid)
{
  systemState = "FAULT";

  controlRequest = 0;


  digitalWrite(
    READY_LED,
    LOW
  );

  digitalWrite(
    FAULT_LED,
    HIGH
  );

  digitalWrite(
    ACTUATOR_LED,
    LOW
  );

  return;
}


// NORMAL
systemState = "READY";


controlRequest =
  (
    apps1Percent
    +
    apps2Percent
```

```
    )
    / 2.0;


  digitalWrite(
    READY_LED,
    HIGH
  );

  digitalWrite(
    FAULT_LED,
    LOW
  );

  digitalWrite(
    ACTUATOR_LED,
    HIGH
  );
}


// ==========================================
// DIAGNOSTICS
// ==========================================

void printStatus()
{
  Serial.println(
    "================================="
  );

  Serial.println(
    "EV SENSOR SAFETY CONTROLLER"
  );


  Serial.print("APPS1 RAW = ");
  Serial.println(adc1);

  Serial.print("APPS2 RAW = ");
  Serial.println(adc2);
```

```
Serial.print("APPS1 = ");
Serial.print(apps1Percent, 1);
Serial.println(" %");

Serial.print("APPS2 = ");
Serial.print(apps2Percent, 1);
Serial.println(" %");


Serial.print("Difference = ");
Serial.print(difference, 1);
Serial.println(" %");


Serial.print("APPS1 RANGE = ");

Serial.println(
  apps1RangeOK
    ? "OK"
    : "FAULT"
);


Serial.print("APPS2 RANGE = ");

Serial.println(
  apps2RangeOK
    ? "OK"
    : "FAULT"
);


Serial.print("PLAUSIBILITY = ");

Serial.println(
  plausibilityOK
    ? "OK"
    : "FAULT"
);


Serial.print("SENSOR VALID = ");
```

```
  Serial.println(
    sensorValid
      ? "YES"
      : "NO"
  );


  Serial.print("CONTROL REQUEST = ");

  Serial.print(
    controlRequest,
    1
  );

  Serial.println(" %");


  Serial.print("SYSTEM = ");

  Serial.println(
    systemState
  );


  Serial.println(
    "==================================="
  );

  Serial.println();
}


// ========================================
// SETUP
// ========================================

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

  analogReadResolution(12);
```

```
  pinMode(
    ENABLE_PIN,
    INPUT_PULLUP
  );


  pinMode(
    READY_LED,
    OUTPUT
  );

  pinMode(
    FAULT_LED,
    OUTPUT
  );

  pinMode(
    ACTUATOR_LED,
    OUTPUT
  );


  // TIM2 = 100 ms
  ControlTimer->setOverflow(
    100000,
    MICROSEC_FORMAT
  );


  ControlTimer->attachInterrupt(
    timerISR
  );


  ControlTimer->resume();
}


// ========================================
// LOOP
// ========================================

void loop()
```

```
{
  if (task100ms)
  {
    task100ms = false;


    processSensors();

    updateControl();


    // Diagnostics = 500 ms
    diagnosticCounter++;


    if (diagnosticCounter >= 5)
    {
      diagnosticCounter = 0;

      printStatus();
    }
  }
}
```

Микроконтроллер датчиктен алынған мәнді электромобильді басқару үшін бірден пайдаланбауы тиіс. Алдымен ол сигналдың электрлік тұрғыдан рұқсат етілген диапазонда екенін тексереді, содан кейін өлшеу нәтижелерін сүзгілейді, екі тәуелсіз арнаның сигналдарын салыстырады және барлық тексерулер сәтті өткеннен кейін ғана басқаруға рұқсат береді.

Екі арнаны тек акселератор педалінің сигналын тексеру үшін ғана емес, басқа да физикалық шамаларды бақылау үшін қолдануға болады.
Бұл жерде екі арна дегеніміз әрқашан бір датчиктің екі шығысы дегенді білдірмейді. Нақты автомобиль жүйелерінде бір физикалық шаманы бақылау үшін:

- бір датчиктің екі сигналдық арнасы;
- екі бөлек датчик;
- немесе екі тәуелсіз өлшеу жолы

қолданылуы мүмкін.

Сондықтан зертханалық жұмыста екі потенциометр бір физикалық шаманы өлшейтін екі тәуелсіз немесе өзара салыстырылатын сигналдық арнаны шартты түрде модельдейді.

Жеке тапсырма

Әр студент екі потенциометрді өз нұсқасына сәйкес бір физикалық шаманы өлшейтін екі тәуелсіз немесе өзара салыстырылатын сигналдық арна ретінде модельдеуі тиіс.

Ескерту: Нақты автомобиль жүйесінде екі арна әртүрлі тәсілмен жүзеге асырылуы мүмкін: бір датчиктің екі сигналдық арнасы, екі бөлек датчик немесе екі тәуелсіз өлшеу жолы. Бұл зертханалық жұмыста олардың барлығы екі потенциометр арқылы шартты түрде модельденеді.

Осылайша студенттер тек акселератор сигналының ғана емес, температура, кернеу, ток, қысым және айналу жиілігі сияқты басқа да маңызды физикалық шамалардың сигналдарын өзара салыстыру және олардың сәйкестігін тексеру принципін зерттейді.

Барлық шекті мәндер оқу мақсатында берілген.

№	Модельденетін сигнал	Физикалық диапазон	Арналар арасындағы максималды айырмашылық
1	Батарея температурасы	−20…80 °C	5 °C
2	Электр қозғалтқышының температурасы	0–150 °C	8 °C
3	Инвертор температурасы	0–120 °C	6 °C
4	Батарея кернеуі	250–450 V	10 V
5	DC-link кернеуі	0–500 V	15 V
6	Батарея тогы	−300…300 A	20 A
7	Салқындату жүйесінің қысымы	0–5 bar	0.3 bar
8	Электр қозғалтқышының айналу жиілігі	0–12000 rpm	300 rpm

Студент:

1. екі ADC арнасын оқуы;
2. ADC мәндерін физикалық шамаға түрлендіруі;
3. Range Checking орындауы;

4. екі арнаның айырмашылығын есептеуі;
5. Plausibility Checking орындауы;
6. Fault Detection орындауы;
7. Fault кезінде басқарушы шығысты өшіруі;
8. Serial Monitor арқылы диагностикалық ақпарат шығаруы тиіс.

Бақылау сұрақтары

1. ADC қандай мақсатта қолданылады?
2. 12-биттік ADC қандай сандық диапазон береді?
3. Неліктен датчиктің жұмыс диапазоны толық 0–4095 диапазонына тең болмауы мүмкін?
4. Range Checking дегеніміз не?
5. Range Checking автомобиль жүйесінде не үшін қажет?
6. Датчик сигналы 0 V-қа жақын болған жағдайда қандай ақау болуы мүмкін?
7. Датчик сигналы қорек кернеуіне жақын болған жағдайда қандай ақау болуы мүмкін?
8. Accelerator Pedal Position Sensor дегеніміз не?
9. APPS деген қысқартудың мағынасы қандай?
10. Redundancy дегеніміз не?
11. Неліктен маңызды датчиктерде бірнеше сигналдық арна қолданылуы мүмкін?
12. Plausibility Checking дегеніміз не?
13. Екі датчик сигналының айырмашылығы қалай анықталады?
14. Бұл зертханалық жұмыста қандай айырмашылық Plausibility Fault тудырады?
15. Неліктен Range Checking және Plausibility Checking екеуі де қажет?
16. Moving Average Filter қандай мақсатта қолданылады?
17. Неліктен Range Checking RAW ADC мәні бойынша орындалады?
18. Fault Detection дегеніміз не?
19. Safe State дегеніміз не?
20. Ақау кезінде неге Control Request нөлге теңестіріледі?
21. ACTUATOR LED зертханалық жұмыста нені модельдейді?
22. Неліктен нақты қуатты атқарушы құрылғы MCU шығысына тікелей қосылмайды?
23. TIM2 бұл зертханалық жұмыста қандай мақсатта қолданылады?
24. Функционалдық қауіпсіздік автомобильдік embedded-жүйелер үшін неге маңызды?