BEÁGYAZOTT RENDSZEREK (C)

3. forduló

BOSCH

A kategória támogatója: Robert Bosch Kft.

Ismertető a feladathoz

A 3.forduló feladatait a hosszú hétvége miatt kivételesen szerda (11.02.) éjfélig tudod megoldani!

Érdemes ebben a fordulóban is játszanod, mert a következő forduló kezdetekor, 11.03-án 18 órától kiosztjuk az 1.-2.-3. fordulóban megszerzett badgeket!

A verseny közben az alábbi teljesítményeket díjazzuk:

- fordulógyőztes
- átlagnál jobb időeredmény
- átlag feletti pontszám
- hibátlan forduló

Szeretnénk rá felhívni figyelmedet, hogy az egyszer megkapott badge-eket nem vonjuk vissza, akkor sem, ha esetleg az adott fordulóban a visszajelzések alapján változások vannak.

Jó játékot!

3.forduló

Gipsz Szabolcs folytatja a robotfűnyíró fejlesztését. Ezúttal a hajtómotorok vezérlését tervezi megvalósítani. Időközben megérkezett az STM32 Nucleo-F103RB kártyája, amit a fűnyíró perifériáinak végleges vezérléséhez rendelt.

A megoldásban a következő adatlapok lesznek a segítségedre:

STM32F103RB adatlap: https://www.st.com/resource/en/datasheet/stm32f103rb.pdf

NUCLEO-F103RB adatlap: https://www.st.com/resource/en/data_brief/nucleo-f103rb.pdf

HAL API dokumentáció: https://www.st.com/resource/en/user_manual/dm00154093-description-of-stm32f1-hal-and-lowlayer-drivers-stmicroelectronics.pdf

PID szabályozó: https://hu.wikipedia.org/wiki/PID_szab%C3%A1lyoz%C3%B3

Felhasznált idő: 00:00/25:00 **Elért pontszám:** 0/32

1. feladat 0/3 pont

Szabolcs differenciálhajtást tervez a fűnyíróhoz, ami azt jelenti, hogy lesz egy jobb és egy bal oldali hajtómotor. Mindkét hajtásban elhelyez egy kvadratúra enkódert, melyek jeleit az STM timerei által támogatott enkóder mód igénybevételével tervezi kezelni. A

motorok teljesítményét az STM timerei által szintén támogatott hardveres PWM jel generálással tervezi szabályozni.

Minimum hány timerre lesz szüksége a fenti célok eléréséhez (számmal add meg)!

Válasz

A helyes válasz:

3

Magyarázat

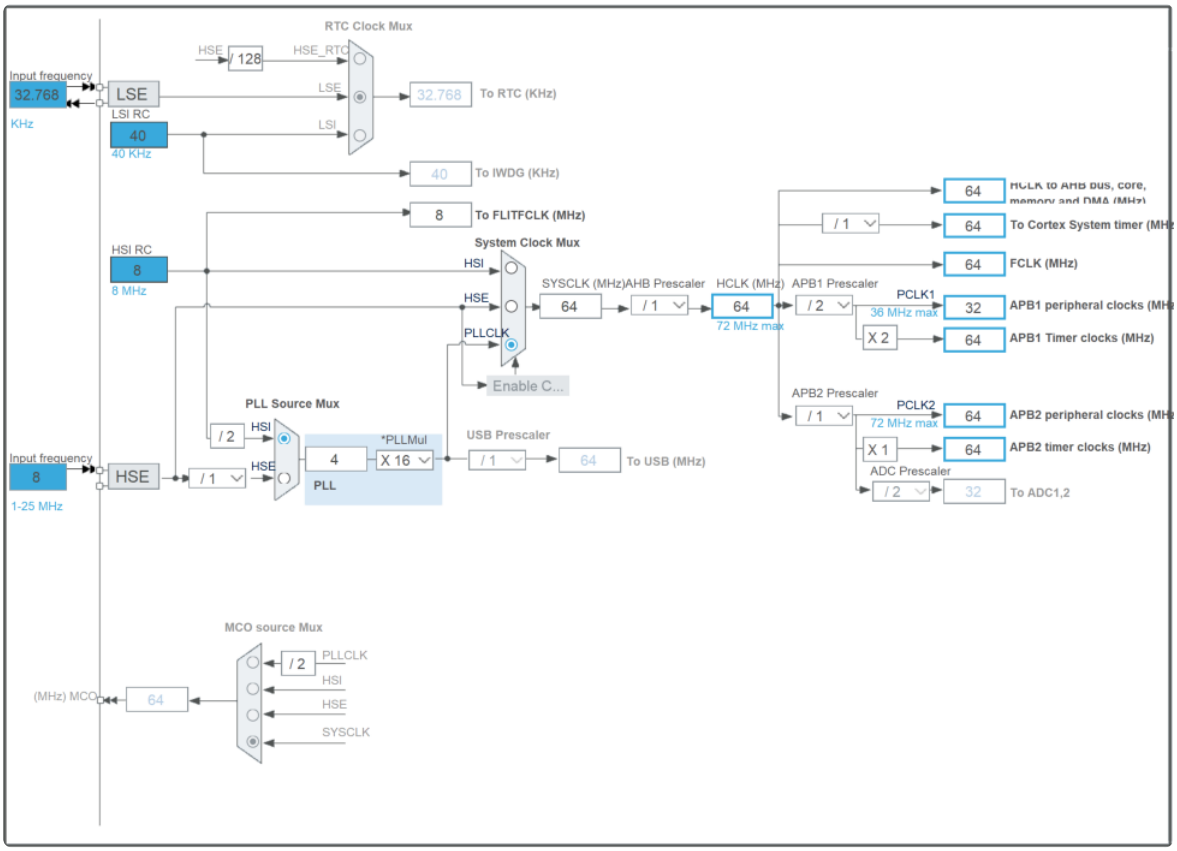
Egy-egy timerre lesz szüksége az enkóderekhez, a PWM jel generáláshoz viszont elegendő egy közös timer, ami több (leggyakrabban 4) csatornán keresztül szimultán képes előállítani egyező frekvenciájú, de egyedi pulzusszélességű jeleket.

2. feladat 0/10 pont

Szabolcs szeretné PWM-mel szabályozni a hajtómotorok teljesítményét. Az STM egyik timerét felhasználva szeretne 10 kHz-es PWM jelet előállítani, aminek kitöltési tényezőjét próbaképp a kártyán lévő nyomógomb segítségével szeretné több lépésben (0%, 25%, 50%, 75%, 100%) változtatni. A PWM jellel egy teljesítményelektronikán keresztül tervezi vezérelni az egyik 12 V-os DC motort. A motor tengelyére szerelt kvadratúra enkóder jeleit a TIM3 timer dolgozza fel, így a PWM generáláshoz a TIM2 timert választja.

Szabolcs picit szétszórt. Egyszer csak azon kapja magát, hogy négy projektet is indított az STM32CubeIDE-ben, és már nem biztos benne, hogy melyik kód működik. **Segíts neki kiválasztani a feladatot helyesen megvalósító kódrészleteket!**

A beállított clock tree-t az alábbi ábra szemlélteti:



Válaszok

- [x] 1.válasz:

```c
#include "main.h"

TIM_HandleTypeDef htim2;
UART_HandleTypeDef huart2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM2_Init();

  GPIO_PinState btn_prevstate = HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  uint8_t pulseWidthPercent = 0;

  while (1)
  {
      if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) != btn_prevstate) {
          if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_SET) {
              pulseWidthPercent += 25;
              if(pulseWidthPercent > 100)
                  pulseWidthPercent = 0;
              htim2.Instance->CCR1 = htim2.Instance->ARR * pulseWidthPercent / 100;
          }
          btn_prevstate = HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin);
      }
      HAL_Delay(20);
  }
}

static void MX_TIM2_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 63;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 99;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  HAL_TIM_MspPostInit(&htim2);
}
```

2.válasz:

```c
#include "main.h"

TIM_HandleTypeDef htim2;
UART_HandleTypeDef huart2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM2_Init();

  GPIO_PinState btn_prevstate = HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  uint8_t pulseWidthPercent = 0;

  while (1)
  {
      if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) != btn_prevstate) {
          if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_SET) {
              pulseWidthPercent += 25;
              if(pulseWidthPercent > 100)
                  pulseWidthPercent = 0;
              htim2.Instance->CCR1 = htim2.Instance->ARR * pulseWidthPercent / 100;
          }
          btn_prevstate = HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin);
      }
      HAL_Delay(20);
  }
}

static void MX_TIM2_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 6400;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 100;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  HAL_TIM_MspPostInit(&htim2);
}
```

3.válasz:

```c
#include "main.h"

TIM_HandleTypeDef htim2;
UART_HandleTypeDef huart2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM2_Init();

  GPIO_PinState btn_prevstate = HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  uint8_t pulseWidthPercent = 0;

  while (1)
  {
      if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) != btn_prevstate) {
          if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_SET) {
              pulseWidthPercent += 25;
              if(pulseWidthPercent > 100)
                  pulseWidthPercent = 0;
              htim2.Instance->CCR1 = htim2.Instance->ARR * pulseWidthPercent / 100;
          }
          btn_prevstate = HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin);
      }
      HAL_Delay(20);
  }
}

static void MX_TIM2_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 6399;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
  }
  HAL_TIM_MspPostInit(&htim2);
}
```

Ez a válasz helyes, de nem jelölted meg.

4.válasz:

```c
#include "main.h"

TIM_HandleTypeDef htim2;
UART_HandleTypeDef huart2;

void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);

int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM2_Init();

  GPIO_PinState btn_prevstate = HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin);
  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
  uint8_t pulseWidthPercent = 0;

  while (1)
  {
      if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) != btn_prevstate) {
          if(HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin) == GPIO_PIN_SET) {
              pulseWidthPercent += 25;
              if(pulseWidthPercent > 100)
                  pulseWidthPercent = 0;
              htim2.Instance->CCR1 = pulseWidthPercent;
          }
          btn_prevstate = HAL_GPIO_ReadPin(B1_GPIO_Port, B1_Pin);
      }
      HAL_Delay(20);
  }
}

static void MX_TIM2_Init(void)
{
  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};

  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 32;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 200;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
  {
    Error_Handler();
```

```
  }
  HAL_TIM_MspPostInit(&htim2);
}
```

Magyarázat

Az első és a harmadik megoldás is jó. A TIM2 az ABP1 buszról kapja az órajelet, ami 64 MHz-es. Ezt a timer prescalerével és a periódusának (AutoReloadRegister határozza meg) hosszával osztva kapjuk meg a PWM jel frekvenciáját. A harmadik megoldás finomabb lépésközt tesz lehetővé a pulzusszélesség beállításánál. Figyeljünk rá, hogy a prescaler és az ARR értéknél is a szükséges faktor értékéből egyet le kell vonni!

A második megoldásnál a PWM jel frekvenciája csak kb. 100 Hz lesz, a negyedik megoldásnál pedig a pulzusszélességet állítjuk be hibásan (a szükségeshez képest fele nagyságúra).

3. feladat 0/4 pont

Szabolcs szeretne sebességszabályozást megvalósítani, amihez egy PID szabályozót akar implementálni az STM32 vezérlőben. Kicsit már utána is olvasott a PID szabályozásnak, de még nem teljesen biztos a dolgában. Segíts neki a helyes válaszok megtalálásában!

Jelöld meg az igaz állításokat!

Válaszok

- [x] Arányos (P) szabályozóval mindig marad egy kis hiba a szabályozott jelben.
  Ez a válasz helyes, de nem jelölted meg.
- [x] Az integráló tag idővel kiküszöböli a P szabályozó által rendszerben hagyott hibát.
  Ez a válasz helyes, de nem jelölted meg.
- [ ] A PI szabályozó gyorsan beállítja a vezérlő jelet, amiben az oszcillációk hamar lecsengenek.
- [x] A differenciáló (D) tag segít a pontos szabályozásban változó sebességű hibajel esetén.
  Ez a válasz helyes, de nem jelölted meg.
- [ ] A PID szabályozó tagjai által adott eredményeket összeszorozva kapjuk a vezérlő jelet.
- [x] A hiba változásának nagysága a hibajel meredekségéből határozható meg.
  Ez a válasz helyes, de nem jelölted meg.

Magyarázat

Az *"A PI szabályozó gyorsan beállítja a vezérlő jelet, amiben az oszcillációk hamar lecsengenek."* válasz hamis: PI szabályozóval gyorsan megközelíthetjük a kívánt állapotot, de a vezérlő jel hosszan fog oszcillálni. Ennek csillapítására hasznos a differenciáló tag.

A *"A PID szabályozó tagjai által adott eredményeket összeszorozva kapjuk a vezérlő jelet."* válasz is hamis: az egyes tagok által szolgáltatott jeleket szummázni kell.

https://hu.wikipedia.org/wiki/PID_szab%C3%A1lyoz%C3%B3

4. feladat 0/15 pont

Szabolcs szeretné a hajtómotorok sebességét szabályozni, amihez egy PID szabályozót implementál az STM-ben. A visszacsatoláshoz már stabilan tudja kezelni az enkódereket, sőt, már a sebesség mérését is megoldotta egy időmérésre használt timer segítségével.
Segíts neki eldönteni, hogy melyik kódrészlet implementálja helyesen a PID szabályozót!

Válasz

1.válasz:

```
int main(void) {
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM2_Init();

  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);

  int target_speed = DEFAULT_TARGET_SPEED;
  int speed_integral = 0;
  int speed_derivative = 0;
  int last_speed_error = 0;

  while (1) {

      int current_speed = getMotorSpeed();
      int speed_error = target_speed - current_speed;
      speed_integral += speed_error;
      speed_derivative = speed_error - last_speed_error;

      control_signal = Kp * speed_error + Ki * speed_integral + Kd * speed_derivative;

      if(control_signal > MAX_PULSE_WIDTH) control_signal = MAX_PULSE_WIDTH;
      else if(control_signal < -MAX_PULSE_WIDTH) control_signal = -MAX_PULSE_WIDTH;

      if(control_signal > 0)
          setMotorDir(FORWARD);
      else if(control_signal < 0)
          setMotorDir(BACKWARD);

      htim2.Instance->CCR1 = abs(control_signal);
     HAL_Delay(1);
  }
}
```

2.válasz:

```c
int main(void) {
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM2_Init();

  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);

  int target_speed = DEFAULT_TARGET_SPEED;
  int speed_integral = 0;
  int speed_derivative = 0;
  int last_speed_error = 0;

  while (1) {

      int current_speed = getMotorSpeed();
      int speed_error = target_speed - current_speed;
      speed_integral += current_speed;
      speed_derivative = speed_error - last_speed_error;

      control_signal = Kp * speed_error + Ki * speed_integral + Kd * speed_derivative;
      last_speed_error = speed_error;

      if(control_signal > MAX_PULSE_WIDTH) control_signal = MAX_PULSE_WIDTH;
      else if(control_signal < -MAX_PULSE_WIDTH) control_signal = -MAX_PULSE_WIDTH;

      if(control_signal > 0)
          setMotorDir(FORWARD);
      else if(control_signal < 0)
          setMotorDir(BACKWARD);

      htim2.Instance->CCR1 = abs(control_signal);
     HAL_Delay(1);
  }
}
```

3.válasz:

```
int main(void) {
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM2_Init();

  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);

  int target_speed = DEFAULT_TARGET_SPEED;
  int speed_integral = 0;
  int speed_derivative = 0;
  int last_speed_error = 0;

  while (1) {

      int current_speed = getMotorSpeed();
      int speed_error = target_speed - current_speed;
      speed_integral += speed_error;
      speed_derivative = speed_error - last_speed_error;

      control_signal = Kp * speed_error + Ki * speed_integral + Kd * speed_derivative;
      last_speed_error = speed_error;

      if(control_signal > MAX_PULSE_WIDTH) control_signal = MAX_PULSE_WIDTH;
      else if(control_signal < -MAX_PULSE_WIDTH) control_signal = -MAX_PULSE_WIDTH;

      if(control_signal > 0)
          setMotorDir(FORWARD);
      else if(control_signal < 0)
          setMotorDir(BACKWARD);

      htim2.Instance->CCR1 = abs(control_signal);
     HAL_Delay(1);
  }
}
```

Ez a válasz helyes, de nem jelölted meg.

4.válasz:

```c
int main(void) {
  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM2_Init();

  HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);

  unsigned int target_speed = DEFAULT_TARGET_SPEED;
  unsigned int speed_integral = 0;
  unsigned int speed_derivative = 0;
  unsigned int last_speed_error = 0;

  while (1) {

      unsigned int current_speed = getMotorSpeed();
      unsigned int speed_error = target_speed - current_speed;
      speed_integral += speed_error;
      speed_derivative = speed_error - last_speed_error;

      control_signal = Kp * speed_error + Ki * speed_integral + Kd * speed_derivative;
      last_speed_error = speed_error;

      if(control_signal > MAX_PULSE_WIDTH) control_signal = MAX_PULSE_WIDTH;
      else if(control_signal < -MAX_PULSE_WIDTH) control_signal = -MAX_PULSE_WIDTH;

      if(control_signal > 0)
          setMotorDir(FORWARD);
      else if(control_signal < 0)
          setMotorDir(BACKWARD);

      htim2.Instance->CCR1 = abs(control_signal);
     HAL_Delay(1);
  }
}
```

Magyarázat

Az első megoldásban nem frissítjük a last_speed_error értékét.

A második megoldásban a speed_integralban az aktuális sebességeket akkumuláljuk az aktuális sebesség hibák helyett.

A harmadik a jó megoldás.

A negyedik megoldásban unsigned int-eket használunk, a szabályozáshoz pedig szükség van a negatív tartományra is.