附录

单片机代码实现

/\* USER CODE BEGIN Header \*/

/\*\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @file : main.c

\* @brief : Main program body

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* @attention

\*

\* Copyright (c) 2023 STMicroelectronics.

\* All rights reserved.

\*

\* This software is licensed under terms that can be found in the LICENSE file

\* in the root directory of this software component.

\* If no LICENSE file comes with this software, it is provided AS-IS.

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\* USER CODE END Header \*/

/\* Includes ------------------------------------------------------------------\*/

#include "main.h"

/\* Private includes ----------------------------------------------------------\*/

/\* USER CODE BEGIN Includes \*/

#include "u8g2.h"

#include <stdio.h>

#include <stdlib.h>

#include <stdarg.h>

#include <math.h>

/\* USER CODE END Includes \*/

/\* Private typedef -----------------------------------------------------------\*/

/\* USER CODE BEGIN PTD \*/

/\* USER CODE END PTD \*/

/\* Private define ------------------------------------------------------------\*/

/\* USER CODE BEGIN PD \*/

/\* USER CODE END PD \*/

/\* Private macro -------------------------------------------------------------\*/

/\* USER CODE BEGIN PM \*/

#define ADC\_ARRAY\_NUM (1024)

#define Kp 5

#define Ki 0.05

#define Kd 2

// #define Kd 0

/\* USER CODE END PM \*/

/\* Private variables ---------------------------------------------------------\*/

SPI\_HandleTypeDef hspi1;

TIM\_HandleTypeDef htim1;

TIM\_HandleTypeDef htim2;

TIM\_HandleTypeDef htim5;

TIM\_HandleTypeDef htim10;

UART\_HandleTypeDef huart1;

/\* USER CODE BEGIN PV \*/

u8g2\_t u8g2;

uint16\_t adcvalue = 0;

uint16\_t adcarray[ADC\_ARRAY\_NUM];

uint32\_t motor\_on = 0;

uint32\_t motor\_forward = 1;

uint32\_t motor\_pwm\_freq = 84;

uint32\_t beep\_freq = 2700;

uint32\_t beep\_on = 0;

uint32\_t key\_1\_tick = 0;

uint32\_t key\_2\_tick = 0;

uint32\_t key\_3\_tick = 0;

uint32\_t key\_4\_tick = 0;

float speed = 0;

int target\_speed = 180;

float motor\_pwm\_duty = 0;

float deviate = 0;

/\* USER CODE END PV \*/

/\* Private function prototypes -----------------------------------------------\*/

void SystemClock\_Config(void);

static void MX\_GPIO\_Init(void);

static void MX\_TIM1\_Init(void);

static void MX\_TIM2\_Init(void);

static void MX\_SPI1\_Init(void);

static void MX\_USART1\_UART\_Init(void);

static void MX\_TIM5\_Init(void);

static void MX\_TIM10\_Init(void);

/\* USER CODE BEGIN PFP \*/

int UART\_printf(UART\_HandleTypeDef \*huart, const char \*fmt, ...);

int u8g2\_printf(u8g2\_t \*u8g2, u8g2\_uint\_t x, u8g2\_uint\_t y, const char \*fmt,

...);

uint8\_t u8x8\_stm32\_gpio\_and\_delay(U8X8\_UNUSED u8x8\_t \*u8x8,

U8X8\_UNUSED uint8\_t msg,

U8X8\_UNUSED uint8\_t arg\_int,

U8X8\_UNUSED void \*arg\_ptr);

uint8\_t u8x8\_byte\_4wire\_hw\_spi(u8x8\_t \*u8x8, uint8\_t msg, uint8\_t arg\_int, void \*arg\_ptr);

/\* USER CODE END PFP \*/

/\* Private user code ---------------------------------------------------------\*/

/\* USER CODE BEGIN 0 \*/

/\* USER CODE END 0 \*/

/\*\*

\* @brief The application entry point.

\* @retval int

\*/

int main(void)

{

/\* USER CODE BEGIN 1 \*/

/\* USER CODE END 1 \*/

/\* MCU Configuration--------------------------------------------------------\*/

/\* Reset of all peripherals, Initializes the Flash interface and the Systick. \*/

HAL\_Init();

/\* USER CODE BEGIN Init \*/

/\* USER CODE END Init \*/

/\* Configure the system clock \*/

SystemClock\_Config();

/\* USER CODE BEGIN SysInit \*/

/\* USER CODE END SysInit \*/

/\* Initialize all configured peripherals \*/

MX\_GPIO\_Init();

MX\_TIM1\_Init();

MX\_TIM2\_Init();

MX\_SPI1\_Init();

MX\_USART1\_UART\_Init();

MX\_TIM5\_Init();

MX\_TIM10\_Init();

/\* USER CODE BEGIN 2 \*/

/\* USER CODE END 2 \*/

/\* Infinite loop \*/

/\* USER CODE BEGIN WHILE \*/

u8g2\_Setup\_ssd1306\_128x64\_noname\_2(&u8g2, U8G2\_R0, u8x8\_byte\_4wire\_hw\_spi, u8x8\_stm32\_gpio\_and\_delay); // init u8g2 structure

u8g2\_InitDisplay(&u8g2); // send init sequence to the display, display is in sleep mode after this,

u8g2\_SetPowerSave(&u8g2, 0); // wake up display

HAL\_TIM\_Encoder\_Start(&htim5, TIM\_CHANNEL\_1);

HAL\_TIM\_Encoder\_Start(&htim5, TIM\_CHANNEL\_2);

UART\_printf(&huart1, "These messages are sent by UART\_printf!\r\n");

UART\_printf(&huart1, "Build on "\_\_TIME\_\_

" "\_\_DATE\_\_

" \r\n");

HAL\_TIM\_Base\_Start\_IT(&htim10);

HAL\_TIM\_PWM\_Start(&htim2, TIM\_CHANNEL\_1);

HAL\_TIM\_PWM\_Start(&htim1, TIM\_CHANNEL\_2);

HAL\_TIM\_PWM\_Start(&htim1, TIM\_CHANNEL\_3);

while (1)

{

if ((fabs(deviate) - 5) >= 0 && motor\_on)

{

beep\_on = 1;

HAL\_GPIO\_WritePin(LED\_R\_GPIO\_Port, LED\_R\_Pin, GPIO\_PIN\_SET);

HAL\_GPIO\_WritePin(LED\_G\_GPIO\_Port, LED\_G\_Pin, GPIO\_PIN\_RESET);

}

else if (motor\_on)

{

beep\_on = 0;

HAL\_GPIO\_WritePin(LED\_G\_GPIO\_Port, LED\_G\_Pin, GPIO\_PIN\_SET);

HAL\_GPIO\_WritePin(LED\_R\_GPIO\_Port, LED\_R\_Pin, GPIO\_PIN\_RESET);

}

else

{

beep\_on = 0;

HAL\_GPIO\_WritePin(LED\_G\_GPIO\_Port, LED\_G\_Pin, GPIO\_PIN\_RESET);

HAL\_GPIO\_WritePin(LED\_R\_GPIO\_Port, LED\_R\_Pin, GPIO\_PIN\_RESET);

}

if (HAL\_GPIO\_ReadPin(KEY\_1\_GPIO\_Port, KEY\_1\_Pin) == GPIO\_PIN\_RESET)

{

if ((HAL\_GetTick() - key\_1\_tick) > 100)

{

motor\_on = motor\_on == 1 ? 0 : 1;

motor\_pwm\_duty = motor\_on == 1 ? motor\_pwm\_duty : 0;

}

// Record the last detection tick

key\_1\_tick = HAL\_GetTick();

}

if (HAL\_GPIO\_ReadPin(KEY\_2\_GPIO\_Port, KEY\_2\_Pin) == GPIO\_PIN\_RESET)

{

if ((HAL\_GetTick() - key\_2\_tick) > 100)

{

motor\_forward = (motor\_forward == 0) ? 1 : 0;

}

// Record the last detection tick

key\_2\_tick = HAL\_GetTick();

}

if (HAL\_GPIO\_ReadPin(KEY\_3\_GPIO\_Port, KEY\_3\_Pin) == GPIO\_PIN\_RESET)

{

if ((HAL\_GetTick() - key\_3\_tick) > 100)

{

// beep\_on = (beep\_on == 0) ? 1 : 0;

if (target\_speed > 20)

target\_speed -= 1;

// Record the last detection tick

key\_3\_tick = HAL\_GetTick();

}

}

if (HAL\_GPIO\_ReadPin(KEY\_4\_GPIO\_Port, KEY\_4\_Pin) == GPIO\_PIN\_RESET)

{

if ((HAL\_GetTick() - key\_4\_tick) > 100)

{

if (target\_speed < 230)

target\_speed += 1;

// Record the last detection tick

key\_4\_tick = HAL\_GetTick();

}

}

if (motor\_on == 1)

{

static float last\_error = 0.0; // 上一次的误差

static float integral = 0.0; // 积分项

float error, derivative, duty\_cycle;

error = target\_speed - speed;

if (motor\_pwm\_duty <= 100)

integral += error; // 计算积分项，同时使用积分遇限削弱法（clamping）实现抗积分饱和

derivative = error - last\_error; // 计算微分项

last\_error = error;

duty\_cycle = Kp \* error + Ki \* integral + Kd \* derivative; // 计算输出值

motor\_pwm\_duty = (duty\_cycle / 10000); // 设置电机占空比

}

\_\_HAL\_TIM\_SetAutoreload(&htim1, ((84000000ul / motor\_pwm\_freq) - 1)); /\* 84MHz/motor\_pwm\_freq is the motor pwm sampling frequency \*/

uint32\_t compare1 = \_\_HAL\_TIM\_GetAutoreload(&htim1);

if (motor\_forward)

{

\_\_HAL\_TIM\_SetCompare(&htim1, TIM\_CHANNEL\_2, (uint32\_t)(compare1 \* (motor\_pwm\_duty / 100.0f)));

\_\_HAL\_TIM\_SetCompare(&htim1, TIM\_CHANNEL\_3, (uint32\_t)compare1 \* 0);

}

else

{

\_\_HAL\_TIM\_SetCompare(&htim1, TIM\_CHANNEL\_3, (uint32\_t)(compare1 \* (motor\_pwm\_duty / 100.0f)));

\_\_HAL\_TIM\_SetCompare(&htim1, TIM\_CHANNEL\_2, (uint32\_t)compare1 \* 0);

}

\_\_HAL\_TIM\_SetAutoreload(&htim2, ((84000000ul / beep\_freq) - 1));

uint32\_t compare2 = \_\_HAL\_TIM\_GetAutoreload(&htim2);

if (beep\_on)

\_\_HAL\_TIM\_SetCompare(&htim2, TIM\_CHANNEL\_1, (uint32\_t)compare2 / 2);

else

\_\_HAL\_TIM\_SetCompare(&htim2, TIM\_CHANNEL\_1, (uint32\_t)compare2 \* 0);

/\* USER CODE END WHILE \*/

/\* USER CODE BEGIN 3 \*/

}

/\* USER CODE END 3 \*/

}

/\*\*

\* @brief System Clock Configuration

\* @retval None

\*/

void SystemClock\_Config(void)

{

RCC\_OscInitTypeDef RCC\_OscInitStruct = {0};

RCC\_ClkInitTypeDef RCC\_ClkInitStruct = {0};

/\*\* Configure the main internal regulator output voltage

\*/

\_\_HAL\_RCC\_PWR\_CLK\_ENABLE();

\_\_HAL\_PWR\_VOLTAGESCALING\_CONFIG(PWR\_REGULATOR\_VOLTAGE\_SCALE2);

/\*\* Initializes the RCC Oscillators according to the specified parameters

\* in the RCC\_OscInitTypeDef structure.

\*/

RCC\_OscInitStruct.OscillatorType = RCC\_OSCILLATORTYPE\_HSI;

RCC\_OscInitStruct.HSIState = RCC\_HSI\_ON;

RCC\_OscInitStruct.HSICalibrationValue = RCC\_HSICALIBRATION\_DEFAULT;

RCC\_OscInitStruct.PLL.PLLState = RCC\_PLL\_ON;

RCC\_OscInitStruct.PLL.PLLSource = RCC\_PLLSOURCE\_HSI;

RCC\_OscInitStruct.PLL.PLLM = 8;

RCC\_OscInitStruct.PLL.PLLN = 84;

RCC\_OscInitStruct.PLL.PLLP = RCC\_PLLP\_DIV2;

RCC\_OscInitStruct.PLL.PLLQ = 4;

if (HAL\_RCC\_OscConfig(&RCC\_OscInitStruct) != HAL\_OK)

{

Error\_Handler();

}

/\*\* Initializes the CPU, AHB and APB buses clocks

\*/

RCC\_ClkInitStruct.ClockType = RCC\_CLOCKTYPE\_HCLK | RCC\_CLOCKTYPE\_SYSCLK | RCC\_CLOCKTYPE\_PCLK1 | RCC\_CLOCKTYPE\_PCLK2;

RCC\_ClkInitStruct.SYSCLKSource = RCC\_SYSCLKSOURCE\_PLLCLK;

RCC\_ClkInitStruct.AHBCLKDivider = RCC\_SYSCLK\_DIV1;

RCC\_ClkInitStruct.APB1CLKDivider = RCC\_HCLK\_DIV2;

RCC\_ClkInitStruct.APB2CLKDivider = RCC\_HCLK\_DIV1;

if (HAL\_RCC\_ClockConfig(&RCC\_ClkInitStruct, FLASH\_LATENCY\_2) != HAL\_OK)

{

Error\_Handler();

}

}

/\*\*

\* @brief SPI1 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_SPI1\_Init(void)

{

/\* USER CODE BEGIN SPI1\_Init 0 \*/

/\* USER CODE END SPI1\_Init 0 \*/

/\* USER CODE BEGIN SPI1\_Init 1 \*/

/\* USER CODE END SPI1\_Init 1 \*/

/\* SPI1 parameter configuration\*/

hspi1.Instance = SPI1;

hspi1.Init.Mode = SPI\_MODE\_MASTER;

hspi1.Init.Direction = SPI\_DIRECTION\_2LINES;

hspi1.Init.DataSize = SPI\_DATASIZE\_8BIT;

hspi1.Init.CLKPolarity = SPI\_POLARITY\_HIGH;

hspi1.Init.CLKPhase = SPI\_PHASE\_2EDGE;

hspi1.Init.NSS = SPI\_NSS\_SOFT;

hspi1.Init.BaudRatePrescaler = SPI\_BAUDRATEPRESCALER\_4;

hspi1.Init.FirstBit = SPI\_FIRSTBIT\_MSB;

hspi1.Init.TIMode = SPI\_TIMODE\_DISABLE;

hspi1.Init.CRCCalculation = SPI\_CRCCALCULATION\_DISABLE;

hspi1.Init.CRCPolynomial = 10;

if (HAL\_SPI\_Init(&hspi1) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN SPI1\_Init 2 \*/

/\* USER CODE END SPI1\_Init 2 \*/

}

/\*\*

\* @brief TIM1 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_TIM1\_Init(void)

{

/\* USER CODE BEGIN TIM1\_Init 0 \*/

/\* USER CODE END TIM1\_Init 0 \*/

TIM\_ClockConfigTypeDef sClockSourceConfig = {0};

TIM\_MasterConfigTypeDef sMasterConfig = {0};

TIM\_OC\_InitTypeDef sConfigOC = {0};

TIM\_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

/\* USER CODE BEGIN TIM1\_Init 1 \*/

/\* USER CODE END TIM1\_Init 1 \*/

htim1.Instance = TIM1;

htim1.Init.Prescaler = 0;

htim1.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim1.Init.Period = 65535;

htim1.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim1.Init.RepetitionCounter = 0;

htim1.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

if (HAL\_TIM\_Base\_Init(&htim1) != HAL\_OK)

{

Error\_Handler();

}

sClockSourceConfig.ClockSource = TIM\_CLOCKSOURCE\_INTERNAL;

if (HAL\_TIM\_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL\_OK)

{

Error\_Handler();

}

if (HAL\_TIM\_PWM\_Init(&htim1) != HAL\_OK)

{

Error\_Handler();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

if (HAL\_TIMEx\_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL\_OK)

{

Error\_Handler();

}

sConfigOC.OCMode = TIM\_OCMODE\_PWM1;

sConfigOC.Pulse = 0;

sConfigOC.OCPolarity = TIM\_OCPOLARITY\_HIGH;

sConfigOC.OCNPolarity = TIM\_OCNPOLARITY\_HIGH;

sConfigOC.OCFastMode = TIM\_OCFAST\_DISABLE;

sConfigOC.OCIdleState = TIM\_OCIDLESTATE\_RESET;

sConfigOC.OCNIdleState = TIM\_OCNIDLESTATE\_RESET;

if (HAL\_TIM\_PWM\_ConfigChannel(&htim1, &sConfigOC, TIM\_CHANNEL\_2) != HAL\_OK)

{

Error\_Handler();

}

if (HAL\_TIM\_PWM\_ConfigChannel(&htim1, &sConfigOC, TIM\_CHANNEL\_3) != HAL\_OK)

{

Error\_Handler();

}

sBreakDeadTimeConfig.OffStateRunMode = TIM\_OSSR\_DISABLE;

sBreakDeadTimeConfig.OffStateIDLEMode = TIM\_OSSI\_DISABLE;

sBreakDeadTimeConfig.LockLevel = TIM\_LOCKLEVEL\_OFF;

sBreakDeadTimeConfig.DeadTime = 0;

sBreakDeadTimeConfig.BreakState = TIM\_BREAK\_DISABLE;

sBreakDeadTimeConfig.BreakPolarity = TIM\_BREAKPOLARITY\_HIGH;

sBreakDeadTimeConfig.AutomaticOutput = TIM\_AUTOMATICOUTPUT\_DISABLE;

if (HAL\_TIMEx\_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN TIM1\_Init 2 \*/

/\* USER CODE END TIM1\_Init 2 \*/

HAL\_TIM\_MspPostInit(&htim1);

}

/\*\*

\* @brief TIM2 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_TIM2\_Init(void)

{

/\* USER CODE BEGIN TIM2\_Init 0 \*/

/\* USER CODE END TIM2\_Init 0 \*/

TIM\_ClockConfigTypeDef sClockSourceConfig = {0};

TIM\_MasterConfigTypeDef sMasterConfig = {0};

TIM\_OC\_InitTypeDef sConfigOC = {0};

/\* USER CODE BEGIN TIM2\_Init 1 \*/

/\* USER CODE END TIM2\_Init 1 \*/

htim2.Instance = TIM2;

htim2.Init.Prescaler = 0;

htim2.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim2.Init.Period = 4294967295;

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();

}

/\* USER CODE BEGIN TIM2\_Init 2 \*/

/\* USER CODE END TIM2\_Init 2 \*/

HAL\_TIM\_MspPostInit(&htim2);

}

/\*\*

\* @brief TIM5 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_TIM5\_Init(void)

{

/\* USER CODE BEGIN TIM5\_Init 0 \*/

/\* USER CODE END TIM5\_Init 0 \*/

TIM\_Encoder\_InitTypeDef sConfig = {0};

TIM\_MasterConfigTypeDef sMasterConfig = {0};

/\* USER CODE BEGIN TIM5\_Init 1 \*/

/\* USER CODE END TIM5\_Init 1 \*/

htim5.Instance = TIM5;

htim5.Init.Prescaler = 0;

htim5.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim5.Init.Period = 4294967295;

htim5.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim5.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

sConfig.EncoderMode = TIM\_ENCODERMODE\_TI12;

sConfig.IC1Polarity = TIM\_ICPOLARITY\_RISING;

sConfig.IC1Selection = TIM\_ICSELECTION\_DIRECTTI;

sConfig.IC1Prescaler = TIM\_ICPSC\_DIV1;

sConfig.IC1Filter = 0;

sConfig.IC2Polarity = TIM\_ICPOLARITY\_RISING;

sConfig.IC2Selection = TIM\_ICSELECTION\_DIRECTTI;

sConfig.IC2Prescaler = TIM\_ICPSC\_DIV1;

sConfig.IC2Filter = 0;

if (HAL\_TIM\_Encoder\_Init(&htim5, &sConfig) != HAL\_OK)

{

Error\_Handler();

}

sMasterConfig.MasterOutputTrigger = TIM\_TRGO\_RESET;

sMasterConfig.MasterSlaveMode = TIM\_MASTERSLAVEMODE\_DISABLE;

if (HAL\_TIMEx\_MasterConfigSynchronization(&htim5, &sMasterConfig) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN TIM5\_Init 2 \*/

/\* USER CODE END TIM5\_Init 2 \*/

}

/\*\*

\* @brief TIM10 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_TIM10\_Init(void)

{

/\* USER CODE BEGIN TIM10\_Init 0 \*/

/\* USER CODE END TIM10\_Init 0 \*/

/\* USER CODE BEGIN TIM10\_Init 1 \*/

/\* USER CODE END TIM10\_Init 1 \*/

htim10.Instance = TIM10;

htim10.Init.Prescaler = 83;

htim10.Init.CounterMode = TIM\_COUNTERMODE\_UP;

htim10.Init.Period = 9999;

htim10.Init.ClockDivision = TIM\_CLOCKDIVISION\_DIV1;

htim10.Init.AutoReloadPreload = TIM\_AUTORELOAD\_PRELOAD\_DISABLE;

if (HAL\_TIM\_Base\_Init(&htim10) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN TIM10\_Init 2 \*/

/\* USER CODE END TIM10\_Init 2 \*/

}

/\*\*

\* @brief USART1 Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_USART1\_UART\_Init(void)

{

/\* USER CODE BEGIN USART1\_Init 0 \*/

/\* USER CODE END USART1\_Init 0 \*/

/\* USER CODE BEGIN USART1\_Init 1 \*/

/\* USER CODE END USART1\_Init 1 \*/

huart1.Instance = USART1;

huart1.Init.BaudRate = 115200;

huart1.Init.WordLength = UART\_WORDLENGTH\_8B;

huart1.Init.StopBits = UART\_STOPBITS\_1;

huart1.Init.Parity = UART\_PARITY\_NONE;

huart1.Init.Mode = UART\_MODE\_TX\_RX;

huart1.Init.HwFlowCtl = UART\_HWCONTROL\_NONE;

huart1.Init.OverSampling = UART\_OVERSAMPLING\_16;

if (HAL\_UART\_Init(&huart1) != HAL\_OK)

{

Error\_Handler();

}

/\* USER CODE BEGIN USART1\_Init 2 \*/

/\* USER CODE END USART1\_Init 2 \*/

}

/\*\*

\* @brief GPIO Initialization Function

\* @param None

\* @retval None

\*/

static void MX\_GPIO\_Init(void)

{

GPIO\_InitTypeDef GPIO\_InitStruct = {0};

/\* USER CODE BEGIN MX\_GPIO\_Init\_1 \*/

/\* USER CODE END MX\_GPIO\_Init\_1 \*/

/\* GPIO Ports Clock Enable \*/

\_\_HAL\_RCC\_GPIOA\_CLK\_ENABLE();

\_\_HAL\_RCC\_GPIOB\_CLK\_ENABLE();

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(OLED\_CS\_GPIO\_Port, OLED\_CS\_Pin, GPIO\_PIN\_RESET);

/\*Configure GPIO pin Output Level \*/

HAL\_GPIO\_WritePin(GPIOB, OLED\_RST\_Pin | OLED\_DC\_Pin | LED\_G\_Pin | LED\_R\_Pin, GPIO\_PIN\_RESET);

/\*Configure GPIO pin : OLED\_CS\_Pin \*/

GPIO\_InitStruct.Pin = OLED\_CS\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(OLED\_CS\_GPIO\_Port, &GPIO\_InitStruct);

/\*Configure GPIO pins : OLED\_RST\_Pin OLED\_DC\_Pin LED\_G\_Pin LED\_R\_Pin \*/

GPIO\_InitStruct.Pin = OLED\_RST\_Pin | OLED\_DC\_Pin | LED\_G\_Pin | LED\_R\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_OUTPUT\_PP;

GPIO\_InitStruct.Pull = GPIO\_NOPULL;

GPIO\_InitStruct.Speed = GPIO\_SPEED\_FREQ\_LOW;

HAL\_GPIO\_Init(GPIOB, &GPIO\_InitStruct);

/\*Configure GPIO pins : KEY\_1\_Pin KEY\_2\_Pin \*/

GPIO\_InitStruct.Pin = KEY\_1\_Pin | KEY\_2\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;

GPIO\_InitStruct.Pull = GPIO\_PULLUP;

HAL\_GPIO\_Init(GPIOB, &GPIO\_InitStruct);

/\*Configure GPIO pins : KEY\_3\_Pin KEY\_4\_Pin \*/

GPIO\_InitStruct.Pin = KEY\_3\_Pin | KEY\_4\_Pin;

GPIO\_InitStruct.Mode = GPIO\_MODE\_INPUT;

GPIO\_InitStruct.Pull = GPIO\_PULLUP;

HAL\_GPIO\_Init(GPIOA, &GPIO\_InitStruct);

/\* USER CODE BEGIN MX\_GPIO\_Init\_2 \*/

/\* USER CODE END MX\_GPIO\_Init\_2 \*/

}

/\* USER CODE BEGIN 4 \*/

uint8\_t u8x8\_stm32\_gpio\_and\_delay(U8X8\_UNUSED u8x8\_t \*u8x8, U8X8\_UNUSED uint8\_t msg, U8X8\_UNUSED uint8\_t arg\_int, U8X8\_UNUSED void \*arg\_ptr)

{

switch (msg)

{

case U8X8\_MSG\_GPIO\_AND\_DELAY\_INIT: // called once during init phase of u8g2/u8x8

HAL\_Delay(1); // init phase of u8x8/u8g2, add a delay of 1 ms

break; // can be used to setup pins

case U8X8\_MSG\_DELAY\_MILLI: // delay arg\_int \* 1 ms

HAL\_Delay(arg\_int);

break;

case U8X8\_MSG\_GPIO\_DC: // DC (data/cmd, A0, register select) pin: Output level in arg\_int

HAL\_GPIO\_WritePin(OLED\_DC\_GPIO\_Port, OLED\_DC\_Pin, arg\_int);

break;

case U8X8\_MSG\_GPIO\_RESET: // Reset pin: Output level in arg\_int

HAL\_GPIO\_WritePin(OLED\_RST\_GPIO\_Port, OLED\_RST\_Pin, arg\_int);

break;

default:

return 0; // a message was received which is not implemented, return 0 to indicate an error

}

return 1;

}

uint8\_t u8x8\_byte\_4wire\_hw\_spi(u8x8\_t \*u8x8, uint8\_t msg, uint8\_t arg\_int,

void \*arg\_ptr)

{

switch (msg)

{

case U8X8\_MSG\_BYTE\_SEND:

HAL\_SPI\_Transmit(&hspi1, (uint8\_t \*)arg\_ptr, arg\_int, HAL\_MAX\_DELAY);

break;

case U8X8\_MSG\_BYTE\_INIT:

break;

case U8X8\_MSG\_BYTE\_SET\_DC:

HAL\_GPIO\_WritePin(OLED\_DC\_GPIO\_Port, OLED\_DC\_Pin, arg\_int);

break;

case U8X8\_MSG\_BYTE\_START\_TRANSFER:

HAL\_GPIO\_WritePin(OLED\_CS\_GPIO\_Port, OLED\_CS\_Pin, GPIO\_PIN\_RESET);

break;

case U8X8\_MSG\_BYTE\_END\_TRANSFER:

HAL\_GPIO\_WritePin(OLED\_CS\_GPIO\_Port, OLED\_CS\_Pin, GPIO\_PIN\_SET);

break;

default:

return 0;

}

return 1;

}

int u8g2\_printf(u8g2\_t \*u8g2, u8g2\_uint\_t x, u8g2\_uint\_t y, const char \*fmt,

...)

{

va\_list ap;

va\_start(ap, fmt);

int length;

char buffer[128];

length = vsnprintf(buffer, 128, fmt, ap);

u8g2\_DrawStr(u8g2, x, y, buffer);

va\_end(ap);

return length;

}

int UART\_printf(UART\_HandleTypeDef \*huart, const char \*fmt, ...)

{

va\_list ap;

va\_start(ap, fmt);

int length;

char buffer[128];

length = vsnprintf(buffer, 128, fmt, ap);

HAL\_UART\_Transmit(huart, (uint8\_t \*)buffer, length, HAL\_MAX\_DELAY);

va\_end(ap);

return length;

}

void HAL\_TIM\_PeriodElapsedCallback(TIM\_HandleTypeDef \*htim)

{

if (htim->Instance == TIM10)

{

static uint16\_t time\_count = 0;

time\_count++;

if (time\_count == 10) // 中断周期为10ms，每10\*10ms触发一次下面的函数

{

static int32\_t lastcount = 0;

int32\_t count = abs(\_\_HAL\_TIM\_GetCounter(&htim5));

speed = (float)(((abs(count - lastcount) / 4.0f) / 11.0f) \* 10.0f \* 60.0f) / 21.3f; // |上次计数值-此次计数值|=编码器在10\*10ms=100ms内的计数 / 一次脉冲下AB两相上升沿+下降沿的计数和=4 / 每圈产生的脉冲信号=10 \* 10\*100ms=1s / 减速比=21.3

int speed\_int = (int)speed;

int speed\_float = (int)(fabs(speed - speed\_int) \* 10);

deviate = (speed - (float)target\_speed) / (float)target\_speed \* 100.0f;

int deviate\_int = (int)deviate;

int deviate\_float = (int)(fabs(deviate - deviate\_int) \* 10);

UART\_printf(&huart1, "%d,%d.%d,%d.%d,%d.%d\n", target\_speed, speed\_int, speed\_float, (int)motor\_pwm\_duty, (int)((motor\_pwm\_duty - (int)motor\_pwm\_duty) \* 100), deviate\_int, deviate\_float);

u8g2\_FirstPage(&u8g2);

do

{

u8g2\_SetFont(&u8g2, u8g2\_font\_samim\_14\_t\_all);

u8g2\_printf(&u8g2, 0, 16, "Target: %d rpm", target\_speed);

char verse\_str = motor\_forward == 1 ? 'F' : 'R';

u8g2\_printf(&u8g2, 112, 16, "%c", verse\_str);

u8g2\_printf(&u8g2, 0, 32, "Speed: %d.%d rpm", speed\_int, speed\_float);

u8g2\_printf(&u8g2, 0, 48, "Duty: %d.%d%%", (int)motor\_pwm\_duty, (int)((motor\_pwm\_duty - (int)motor\_pwm\_duty) \* 100));

u8g2\_printf(&u8g2, 0, 64, "Deviation: %d.%d%%", deviate\_int, deviate\_float);

} while (u8g2\_NextPage(&u8g2));

lastcount = count;

time\_count = 0;

}

}

}

/\* USER CODE END 4 \*/

/\*\*

\* @brief This function is executed in case of error occurrence.

\* @retval None

\*/

void Error\_Handler(void)

{

/\* USER CODE BEGIN Error\_Handler\_Debug \*/

/\* User can add his own implementation to report the HAL error return state \*/

\_\_disable\_irq();

while (1)

{

}

/\* USER CODE END Error\_Handler\_Debug \*/

}

#ifdef USE\_FULL\_ASSERT

/\*\*

\* @brief Reports the name of the source file and the source line number

\* where the assert\_param error has occurred.

\* @param file: pointer to the source file name

\* @param line: assert\_param error line source number

\* @retval None

\*/

void assert\_failed(uint8\_t \*file, uint32\_t line)

{

/\* USER CODE BEGIN 6 \*/

/\* User can add his own implementation to report the file name and line number,

ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) \*/

/\* USER CODE END 6 \*/

}

#endif /\* USE\_FULL\_ASSERT \*/