```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 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 <stdio.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 VOORUIT (GPIOB->BSRR = 0xF800B000);
#define ACHTERUIT (GPIOB->BSRR = 0xF8006800);
#define LINKS (GPIOB->BSRR = 0xF800A800);
#define RECHTS (GPIOB->BSRR = 0xF8007000);
#define UIT (GPIOB->BSRR = 0xF8000000);
#define NORMAL_SPEED 255
#define DOCK_SPEED 145
#define IR_SENSOR_COUNT 3
#define IR_SENSOR_DISTANCE 1500
#define IR_MID_SENSOR_DISTANCE 2200
#define ACHTERUIT_DELAY 500
#define DRAAI_DELAY 1500
#define LANG_DRAAI_DELAY 3000
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
ADC_HandleTypeDef hadc;

TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim6;

UART_HandleTypeDef huart1;

/* USER CODE BEGIN PV */
uint8_t sample = 0;
uint8_t data = 0;
```

```c
uint8_t data_ready = 0;
uint8_t data_beacon = 0;
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_ADC_Init(void);
static void MX_TIM2_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_TIM6_Init(void);
/* USER CODE BEGIN PFP */
int _write(int, char *, int);
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *);
void HAL_GPIO_EXTI_Callback(uint16_t);
void readAdc(uint32_t[]);
uint8_t driveDock(uint8_t, uint32_t[], uint8_t *, uint8_t *);
void driveNormal(uint32_t[]);
/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
int _write(int file, char *ptr, int len)
{
    for(int i = 0; i < len; i++){
        if(ptr[i]=='\n'){
            HAL_UART_Transmit(&huart1, (uint8_t*)"\r", 1, HAL_MAX_DELAY);
        }
        HAL_UART_Transmit(&huart1, (uint8_t*)&ptr[i], 1, HAL_MAX_DELAY);
    }
    return len;
}
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
    //HAL_GPIO_TogglePin(PIN_GPIO_Port, PIN_Pin);
    sample = !HAL_GPIO_ReadPin(IR_GPIO_Port, IR_Pin); //MSB eerst
    data = (data << 1) | sample; //actief lage pin => bit toggelen
    HAL_TIM_Base_Stop_IT(&htim6);
    __HAL_TIM_SET_COUNTER(&htim6, 0);
    if ((data == 0xa4) || (data == 0xa8) || (data == 0xac))
    {
        data_ready = data;
    }
    if (data == 0xa1)
    {
        data_beacon = data;
    }
}
void HAL_GPIO_EXTI_Callback(uint16_t IR_EXTI_IRQn)
{
    HAL_TIM_Base_Start_IT(&htim6);
}
void readAdc(uint32_t result[])
{
    ADC_ChannelConfTypeDef sConfig = {0};

    sConfig.Channel = ADC_CHANNEL_0;
    HAL_ADC_ConfigChannel(&hadc, &sConfig);
    HAL_ADC_Start(&hadc);
    HAL_ADC_PollForConversion(&hadc, 1);
    result[0] = HAL_ADC_GetValue(&hadc);

    sConfig.Channel = ADC_CHANNEL_1;
    HAL_ADC_ConfigChannel(&hadc, &sConfig);
    HAL_ADC_Start(&hadc);
```

```
129     HAL_ADC_PollForConversion(&hadc, 1);
130     result[2] = HAL_ADC_GetValue(&hadc);
131
132     sConfig.Channel = ADC_CHANNEL_2;
133     HAL_ADC_ConfigChannel(&hadc, &sConfig);
134     HAL_ADC_Start(&hadc);
135     HAL_ADC_PollForConversion(&hadc, 1);
136     result[1] = HAL_ADC_GetValue(&hadc);
137
138     printf("RECHTS: %ld, MIDDEN: %ld, LINKS: %ld\n", result[0], result[2],
    result[1]);
139 }
140 uint8_t driveDock(uint8_t state_dock, uint32_t adc_values[], uint8_t *drive_dock,
    uint8_t *detect_dock)
141 {
142     switch (state_dock)
143     {
144     case 0:
145         htim2.Instance->CCR1 = DOCK_SPEED; //Duty Cycle op 50% => led 38kHz
146         htim2.Instance->CCR3 = DOCK_SPEED; //Duty Cycle op 50% => led 38kHz
147         readAdc(adc_values);
148         driveNormal(adc_values);
149
150         if (data_ready == 0xac)
151         {
152             (*detect_dock)++;
153         }
154
155         if (*detect_dock == 1) //150 ms delay
156         {
157             state_dock = 1;
158         }
159         break;
160     case 1:
161         if (data_ready == 0xa4)
162         {
163             VOORUIT;
164             htim2.Instance->CCR1 = 0; //Duty Cycle op 50% => led 38kHz
165             htim2.Instance->CCR3 = DOCK_SPEED; //Duty Cycle op 50% => led 38kHz
166         }
167         else if (data_ready == 0xa8)
168         {
169             VOORUIT;
170             htim2.Instance->CCR1 = DOCK_SPEED; //Duty Cycle op 50% => led 38kHz
171             htim2.Instance->CCR3 = 0; //Duty Cycle op 50% => led 38kHz
172         }
173         else if (data_ready == 0xac)
174         {
175             VOORUIT;
176             htim2.Instance->CCR1 = DOCK_SPEED; //Duty Cycle op 50% => led 38kHz
177             htim2.Instance->CCR3 = DOCK_SPEED; //Duty Cycle op 50% => led 38kHz
178         }
179
180         readAdc(adc_values);
181         if (adc_values[2] > IR_SENSOR_DISTANCE)
182         {
183             state_dock = 2;
184         }
185         break;
186     case 2:
187         UIT;
188         data_ready = 0;
189         data_beacon = 0;
190         if (!HAL_GPIO_ReadPin(DRUK0_GPIO_Port, DRUK0_Pin))
```

```
191         {
192             while (!HAL_GPIO_ReadPin(DRUK0_GPIO_Port, DRUK0_Pin));
193             ACHTERUIT;
194             htim2.Instance->CCR1 = DOCK_SPEED; //Duty Cycle op 50% => led 38kHz
195             htim2.Instance->CCR3 = DOCK_SPEED; //Duty Cycle op 50% => led 38kHz
196             HAL_Delay(ACHTERUIT_DELAY);
197             RECHTS;
198             HAL_Delay(DRAAI_DELAY);
199             *drive_dock = 0;
200             *detect_dock = 0;
201             state_dock = 0;
202         }
203         break;
204     }
205     return state_dock;
206 }
207 void driveNormal(uint32_t adc_values[])
208 {
209     if (HAL_GPIO_ReadPin(DRUK4_GPIO_Port, DRUK4_Pin) == 0) //bumper links
210     {
211         ACHTERUIT;
212         HAL_Delay(ACHTERUIT_DELAY);
213         LINKS;
214         HAL_Delay(DRAAI_DELAY);
215     }
216     else if (HAL_GPIO_ReadPin(DRUK2_GPIO_Port, DRUK2_Pin) == 0) //bumper rechts
217     {
218         ACHTERUIT;
219         HAL_Delay(ACHTERUIT_DELAY);
220         RECHTS;
221         HAL_Delay(DRAAI_DELAY);
222     }
223     else if (HAL_GPIO_ReadPin(DRUK1_GPIO_Port, DRUK1_Pin) == 0) //bumper rechts
224     {
225         ACHTERUIT;
226         HAL_Delay(ACHTERUIT_DELAY);
227         LINKS;
228         HAL_Delay(LANG_DRAAI_DELAY);
229     }
230
231     if (adc_values[0] > IR_SENSOR_DISTANCE && adc_values[1] < IR_SENSOR_DISTANCE)
232     {
233         LINKS;
234         //VOORUIT;
235         //htim2.Instance->CCR1 = NORMAL_SPEED; //Duty Cycle op 50% => led 38kHz
236         //htim2.Instance->CCR3 = 0; //Duty Cycle op 50% => led 38kHz
237     }
238     else if (adc_values[1] > IR_SENSOR_DISTANCE && adc_values[0] <
    IR_SENSOR_DISTANCE)
239     {
240         RECHTS;
241         //VOORUIT;
242         //htim2.Instance->CCR1 = 0; //Duty Cycle op 50% => led 38kHz
243         //htim2.Instance->CCR3 = NORMAL_SPEED; //Duty Cycle op 50% => led 38kHz
244     }
245     else if (adc_values[0] > IR_MID_SENSOR_DISTANCE && adc_values[1] >
    IR_MID_SENSOR_DISTANCE)
246     {
247         ACHTERUIT;
248         HAL_Delay(ACHTERUIT_DELAY);
249         if (adc_values[0] > adc_values[1])
250         {
251             LINKS;
252             HAL_Delay(DRAAI_DELAY);
```

```
253         }
254         else
255         {   RECHTS;
256             HAL_Delay(DRAAI_DELAY);
257         }
258     }
259     else if (adc_values[2] > IR_MID_SENSOR_DISTANCE)
260     {
261         ACHTERUIT;
262         HAL_Delay(ACHTERUIT_DELAY);
263         if (adc_values[0] > adc_values[1])
264         {
265             LINKS;
266             HAL_Delay(DRAAI_DELAY);
267         }
268         else
269         {
270             RECHTS;
271             HAL_Delay(DRAAI_DELAY);
272         }
273     }
274     else
275     {
276         VOORUIT;
277         //htim2.Instance->CCR1 = NORMAL_SPEED; //Duty Cycle op 50% => led 38kHz
278         //htim2.Instance->CCR3 = NORMAL_SPEED; //Duty Cycle op 50% => led 38kHz
279     }
280 }
281 /* USER CODE END 0 */
282
283 /**
284   * @brief  The application entry point.
285   * @retval int
286   */
287 int main void
288 {
289
290   /* USER CODE BEGIN 1 */
291     uint32_t adc_values[IR_SENSOR_COUNT];
292     uint8_t drive_dock = 0;
293     uint8_t state_dock = 0;
294     uint8_t detect_dock = 0;
295   /* USER CODE END 1 */
296
297   /* MCU Configuration--------------------------------------------------------*/
298
299   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
300   HAL_Init();
301
302   /* USER CODE BEGIN Init */
303
304   /* USER CODE END Init */
305
306   /* Configure the system clock */
307   SystemClock_Config();
308
309   /* USER CODE BEGIN SysInit */
310
311   /* USER CODE END SysInit */
312
313   /* Initialize all configured peripherals */
314   MX_GPIO_Init();
315   MX_ADC_Init();
316   MX_TIM2_Init();
```

```
317   MX_USART1_UART_Init();
318   MX_TIM6_Init();
319   /* USER CODE BEGIN 2 */
320   HAL_GPIO_WritePin(IR0_GPIO_Port, IR0_Pin, 1);
321   HAL_GPIO_WritePin(IR1_GPIO_Port, IR1_Pin, 1);
322   HAL_GPIO_WritePin(IR2_GPIO_Port, IR2_Pin, 1);
323   HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
324   HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_3);
325   HAL_Delay(50); //sensors moeten opstarten
326   /* USER CODE END 2 */
327
328   /* Infinite loop */
329   /* USER CODE BEGIN WHILE */
330   while (1)
331   {
332     /* USER CODE END WHILE */
333
334     /* USER CODE BEGIN 3 */
335       switch (drive_dock)
336       {
337       case 0:
338           if (!HAL_GPIO_ReadPin(DRUK0_GPIO_Port, DRUK0_Pin))
339           {
340               while (!HAL_GPIO_ReadPin(DRUK0_GPIO_Port, DRUK0_Pin));
341               drive_dock ^= 1;
342           }
343           HAL_GPIO_WritePin(STOF_GPIO_Port, STOF_Pin, 1);
344           htim2.Instance->CCR1 = NORMAL_SPEED; //Duty Cycle op 50% => led 38kHz
345           htim2.Instance->CCR3 = NORMAL_SPEED; //Duty Cycle op 50% => led 38kHz
346           readAdc(adc_values);
347           printf("l: %d, m: %d, r: %d, ", HAL_GPIO_ReadPin(DRUK4_GPIO_Port,
    DRUK4_Pin), HAL_GPIO_ReadPin(DRUK1_GPIO_Port, DRUK1_Pin),
    HAL_GPIO_ReadPin(DRUK2_GPIO_Port, DRUK2_Pin));
348           driveNormal(adc_values);
349           break;
350       case 1:
351           HAL_GPIO_WritePin(STOF_GPIO_Port, STOF_Pin, 0);
352           printf("data: %x, %x, %d\n", data_ready, data_beacon, state_dock);
353           state_dock = driveDock(state_dock, adc_values, &drive_dock,
    &detect_dock);
354           break;
355       }
356   }
357   /* USER CODE END 3 */
358 }
359
360 /**
361   * @brief System Clock Configuration
362   * @retval None
363   */
364 void SystemClock_Config(void)
365 {
366   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
367   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
368   RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
369
370   /** Configure the main internal regulator output voltage
371   */
372   __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
373
374   /** Initializes the RCC Oscillators according to the specified parameters
375   * in the RCC_OscInitTypeDef structure.
376   */
377   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
```

```
378   RCC_OscInitStruct.HSEState = RCC_HSE_BYPASS;
379   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_NONE;
380   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
381   {
382     Error_Handler();
383   }
384
385   /** Initializes the CPU, AHB and APB buses clocks
386   */
387   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
388                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
389   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_HSE;
390   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
391   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
392   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
393
394   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
395   {
396     Error_Handler();
397   }
398   PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART1;
399   PeriphClkInit.Usart1ClockSelection = RCC_USART1CLKSOURCE_PCLK2;
400   if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
401   {
402     Error_Handler();
403   }
404 }
405
406 /**
407   * @brief ADC Initialization Function
408   * @param None
409   * @retval None
410   */
411 static void MX_ADC_Init(void)
412 {
413
414   /* USER CODE BEGIN ADC_Init 0 */
415
416   /* USER CODE END ADC_Init 0 */
417
418   ADC_ChannelConfTypeDef sConfig = {0};
419
420   /* USER CODE BEGIN ADC_Init 1 */
421
422   /* USER CODE END ADC_Init 1 */
423
424   /** Configure the global features of the ADC (Clock, Resolution, Data Alignment
      and number of conversion)
425   */
426   hadc.Instance = ADC1;
427   hadc.Init.OversamplingMode = DISABLE;
428   hadc.Init.ClockPrescaler = ADC_CLOCK_SYNC_PCLK_DIV2;
429   hadc.Init.Resolution = ADC_RESOLUTION_12B;
430   hadc.Init.SamplingTime = ADC_SAMPLETIME_1CYCLE_5;
431   hadc.Init.ScanConvMode = ADC_SCAN_DIRECTION_FORWARD;
432   hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
433   hadc.Init.ContinuousConvMode = DISABLE;
434   hadc.Init.DiscontinuousConvMode = ENABLE;
435   hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
436   hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
437   hadc.Init.DMAContinuousRequests = DISABLE;
438   hadc.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
439   hadc.Init.Overrun = ADC_OVR_DATA_PRESERVED;
440   hadc.Init.LowPowerAutoWait = DISABLE;
```

```
441   hadc.Init.LowPowerFrequencyMode = DISABLE;
442   hadc.Init.LowPowerAutoPowerOff = DISABLE;
443   if (HAL_ADC_Init(&hadc) != HAL_OK)
444   {
445     Error_Handler();
446   }
447
448   /** Configure for the selected ADC regular channel to be converted.
449   */
450   sConfig.Channel = ADC_CHANNEL_0;
451   sConfig.Rank = ADC_RANK_CHANNEL_NUMBER;
452   if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
453   {
454     Error_Handler();
455   }
456
457   /** Configure for the selected ADC regular channel to be converted.
458   */
459   sConfig.Channel = ADC_CHANNEL_1;
460   if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
461   {
462     Error_Handler();
463   }
464
465   /** Configure for the selected ADC regular channel to be converted.
466   */
467   sConfig.Channel = ADC_CHANNEL_2;
468   if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
469   {
470     Error_Handler();
471   }
472   /* USER CODE BEGIN ADC_Init 2 */
473
474   /* USER CODE END ADC_Init 2 */
475
476 }
477
478 /**
479   * @brief TIM2 Initialization Function
480   * @param None
481   * @retval None
482   */
483 static void MX_TIM2_Init(void)
484 {
485
486   /* USER CODE BEGIN TIM2_Init 0 */
487
488   /* USER CODE END TIM2_Init 0 */
489
490   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
491   TIM_MasterConfigTypeDef sMasterConfig = {0};
492   TIM_OC_InitTypeDef sConfigOC = {0};
493
494   /* USER CODE BEGIN TIM2_Init 1 */
495
496   /* USER CODE END TIM2_Init 1 */
497   htim2.Instance = TIM2;
498   htim2.Init.Prescaler = 93;
499   htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
500   htim2.Init.Period = 255;
501   htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
502   htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
503   if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
504   {
```

```
505     Error_Handler();
506   }
507   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
508   if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
509   {
510     Error_Handler();
511   }
512   if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
513   {
514     Error_Handler();
515   }
516   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
517   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
518   if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
519   {
520     Error_Handler();
521   }
522   sConfigOC.OCMode = TIM_OCMODE_PWM1;
523   sConfigOC.Pulse = 0;
524   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
525   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
526   if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
527   {
528     Error_Handler();
529   }
530   if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
531   {
532     Error_Handler();
533   }
534   /* USER CODE BEGIN TIM2_Init 2 */
535
536   /* USER CODE END TIM2_Init 2 */
537   HAL_TIM_MspPostInit(&htim2);
538
539 }
540
541 /**
542   * @brief TIM6 Initialization Function
543   * @param None
544   * @retval None
545   */
546 static void MX_TIM6_Init(void)
547 {
548
549   /* USER CODE BEGIN TIM6_Init 0 */
550
551   /* USER CODE END TIM6_Init 0 */
552
553   TIM_MasterConfigTypeDef sMasterConfig = {0};
554
555   /* USER CODE BEGIN TIM6_Init 1 */
556
557   /* USER CODE END TIM6_Init 1 */
558   htim6.Instance = TIM6;
559   htim6.Init.Prescaler = 192;
560   htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
561   htim6.Init.Period = 255;
562   htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
563   if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
564   {
565     Error_Handler();
566   }
567   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
568   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
```

```
569   if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) != HAL_OK)
570   {
571     Error_Handler();
572   }
573   /* USER CODE BEGIN TIM6_Init 2 */
574
575   /* USER CODE END TIM6_Init 2 */
576
577 }
578
579 /**
580   * @brief USART1 Initialization Function
581   * @param None
582   * @retval None
583   */
584 static void MX_USART1_UART_Init(void)
585 {
586
587   /* USER CODE BEGIN USART1_Init 0 */
588
589   /* USER CODE END USART1_Init 0 */
590
591   /* USER CODE BEGIN USART1_Init 1 */
592
593   /* USER CODE END USART1_Init 1 */
594   huart1.Instance = USART1;
595   huart1.Init.BaudRate = 115200;
596   huart1.Init.WordLength = UART_WORDLENGTH_8B;
597   huart1.Init.StopBits = UART_STOPBITS_1;
598   huart1.Init.Parity = UART_PARITY_NONE;
599   huart1.Init.Mode = UART_MODE_TX_RX;
600   huart1.Init.HwFlowCtl = UART_HWCONTROL_NONE;
601   huart1.Init.OverSampling = UART_OVERSAMPLING_16;
602   huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
603   huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
604   if (HAL_UART_Init(&huart1) != HAL_OK)
605   {
606     Error_Handler();
607   }
608   /* USER CODE BEGIN USART1_Init 2 */
609
610   /* USER CODE END USART1_Init 2 */
611
612 }
613
614 /**
615   * @brief GPIO Initialization Function
616   * @param None
617   * @retval None
618   */
619 static void MX_GPIO_Init(void)
620 {
621   GPIO_InitTypeDef GPIO_InitStruct = {0};
622 /* USER CODE BEGIN MX_GPIO_Init_1 */
623 /* USER CODE END MX_GPIO_Init_1 */
624
625   /* GPIO Ports Clock Enable */
626   __HAL_RCC_GPIOH_CLK_ENABLE();
627   __HAL_RCC_GPIOA_CLK_ENABLE();
628   __HAL_RCC_GPIOB_CLK_ENABLE();
629
630   /*Configure GPIO pin Output Level */
631   HAL_GPIO_WritePin(STATUS_GPIO_Port, STATUS_Pin, GPIO_PIN_RESET);
632
```

```
633   /*Configure GPIO pin Output Level */
634   HAL_GPIO_WritePin(GPIOA, IR1_Pin|IR2_Pin|IR3_Pin|IR4_Pin
635                           |IR5_Pin|IR0_Pin, GPIO_PIN_RESET);
636
637   /*Configure GPIO pin Output Level */
638   HAL_GPIO_WritePin(GPIOB, BIN2_Pin|BIN1_Pin|STBY_Pin|AIN2_Pin
639                           |AIN1_Pin|STOF_Pin, GPIO_PIN_RESET);
640
641   /*Configure GPIO pin : STATUS_Pin */
642   GPIO_InitStruct.Pin = STATUS_Pin;
643   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
644   GPIO_InitStruct.Pull = GPIO_NOPULL;
645   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
646   HAL_GPIO_Init(STATUS_GPIO_Port, &GPIO_InitStruct);
647
648   /*Configure GPIO pins : IR1_Pin IR2_Pin IR3_Pin IR4_Pin
649                            IR5_Pin IR0_Pin */
650   GPIO_InitStruct.Pin = IR1_Pin|IR2_Pin|IR3_Pin|IR4_Pin
651                           |IR5_Pin|IR0_Pin;
652   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
653   GPIO_InitStruct.Pull = GPIO_NOPULL;
654   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
655   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
656
657   /*Configure GPIO pins : DRUK5_Pin DRUK4_Pin DRUK3_Pin DRUK2_Pin
658                            DRUK1_Pin */
659   GPIO_InitStruct.Pin = DRUK5_Pin|DRUK4_Pin|DRUK3_Pin|DRUK2_Pin
660                           |DRUK1_Pin;
661   GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
662   GPIO_InitStruct.Pull = GPIO_PULLUP;
663   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
664
665   /*Configure GPIO pins : BIN2_Pin BIN1_Pin STBY_Pin AIN2_Pin
666                            AIN1_Pin STOF_Pin */
667   GPIO_InitStruct.Pin = BIN2_Pin|BIN1_Pin|STBY_Pin|AIN2_Pin
668                           |AIN1_Pin|STOF_Pin;
669   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
670   GPIO_InitStruct.Pull = GPIO_NOPULL;
671   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
672   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
673
674   /*Configure GPIO pin : DRUK0_Pin */
675   GPIO_InitStruct.Pin = DRUK0_Pin;
676   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
677   GPIO_InitStruct.Pull = GPIO_PULLUP;
678   HAL_GPIO_Init(DRUK0_GPIO_Port, &GPIO_InitStruct);
679
680   /*Configure GPIO pin : IR_Pin */
681   GPIO_InitStruct.Pin = IR_Pin;
682   GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
683   GPIO_InitStruct.Pull = GPIO_NOPULL;
684   HAL_GPIO_Init(IR_GPIO_Port, &GPIO_InitStruct);
685
686   /* EXTI interrupt init*/
687   HAL_NVIC_SetPriority(EXTI0_1_IRQn, 0, 0);
688   HAL_NVIC_EnableIRQ(EXTI0_1_IRQn);
689
690   HAL_NVIC_SetPriority(EXTI2_3_IRQn, 0, 0);
691   HAL_NVIC_EnableIRQ(EXTI2_3_IRQn);
692
693   HAL_NVIC_SetPriority(EXTI4_15_IRQn, 0, 0);
694   HAL_NVIC_EnableIRQ(EXTI4_15_IRQn);
695
696 /* USER CODE BEGIN MX_GPIO_Init_2 */
```

```
697 /* USER CODE END MX_GPIO_Init_2 */
698 }
699
700 /* USER CODE BEGIN 4 */
701
702 /* USER CODE END 4 */
703
704 /**
705   * @brief  This function is executed in case of error occurrence.
706   * @retval None
707   */
708 void Error_Handler(void)
709 {
710   /* USER CODE BEGIN Error_Handler_Debug */
711   /* User can add his own implementation to report the HAL error return state */
712   __disable_irq();
713   while (1)
714   {
715   }
716   /* USER CODE END Error_Handler_Debug */
717 }
718
719 #ifdef  USE_FULL_ASSERT
720 /**
721   * @brief  Reports the name of the source file and the source line number
722   *         where the assert_param error has occurred.
723   * @param  file: pointer to the source file name
724   * @param  line: assert_param error line source number
725   * @retval None
726   */
727 void assert_failed(uint8_t *file, uint32_t line)
728 {
729   /* USER CODE BEGIN 6 */
730   /* User can add his own implementation to report the file name and line number,
731      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
732   /* USER CODE END 6 */
733 }
734 #endif /* USE_FULL_ASSERT */
735
```